IB Math Analysis and Approaches (HL)

Extended Essay

May 2024 Session

The Mathematics Techniques of Camera Calibration

Research Question: What are the strategies and mathematical techniques employed in camera calibration to develop precise and accurate camera models?

Word Count: 3988 words

Contents

1 Introduction

Camera calibration, also known as camera resectioning, is the process of determining the intrinsic and extrinsic parameters of a camera. The intrinsic parameters deal with the camera's internal characteristics, while the extrinsic parameters describe its position and orientation in the world. The knowledge of the accurate values of these values parameters are essential, as it enables us to create a mathematical model which describes how a camera projects 3D points from a scene onto the 2D image it captures. The importance of a well-calibrated camera becomes very apparent in photogrammetric applications, where precise measurements of 3-dimensional physical objects are derived from photographic images.

Photogrammetry is the science of obtaining accurate measurements of 3-dimensional physical objects through photographic imagery. Photogrammetry was first employed by Prussian architect Albrecht Meydenbauer in the 1860s, who used photogrammetric techniques to create some of the most detailed topographic plans and elevations drawings[1]. Today, photogrammetric techniques are used in a multitude of applications spanning diverse fields, including but not limited to: 3D-model generation, computer vision, topographical mapping, medical imaging, and forensic analysis.

While camera calibration is essential in ensuring the accuracy of photogrammetric applications, it itself also relies on these very same photogrammetric techniques in order to estimate these parameters. In essence, the developments of photogrammetry and camera calibration are closely intertwined, underscoring the essential relationship between photogrammetry and camera calibration.

1.1 Problem Statement

While manufacturers of cameras often report parameters of cameras, such as the nominal focal length and pixel sizes of their camera sensor, these figures are typically approximations which can vary from camera to camera, particularly in consumer-grade cameras. As such, the use of these estimates by manufacturers are unsuitable in developing camera models for

[1] Joerg Albertz, "A Look Back; 140 Years of Photogrammetry," *Journal of the American Society for Photogrammetry and Remote Sensing* 73, no. 5 (May 2007): 1, accessed September 9, 2023, `https://www.asprs.org/wp-content/uploads/pers/2007journal/may/lookback.pdf`.

applications requiring high accuracy. Combined with the potential for manufacturing defects as well as unknown lens distortion coefficients further necessitates the need for a reliable method for determining the parameters of a camera.

Camera calibration emerges as the answer to these problems, allowing us to create very accurate models for the camera as well as generate estimates for its parameters. As such, it is important that we understand the various techniques and strategies use in camera calibration, and how they can be applied to ensure high accuracy.

2 Approach

There are a multitude of approaches one could take to calibrate cameras, and the choice of strategy significantly influences the complexity of the mathematical techniques required. The diversity in these approaches stems from the inherent complexity of cameras and the need for adaptable methods to address specific goals and constraints of each application. Notably, the accuracy of the calibration is significantly influenced by how the camera model is constructed, as well as the type of calibration used[2].

2.1 Camera Model

A camera model is a projection model which approximates the function of a camera by describing a mathematical relationship between points in 3D space and its projection onto the sensor grid of the camera. In order to construct such a model, we must first understand the general workings of a camera.

The modern lens camera is highly sophisticated, built with an array of complex mechanisms and a wide range of features such as zoom and autofocus. However, we only need to focus on its three principal elements critical to image projection: the lens, the aperture, and the sensor grid (CCD).

- **Lens** – Focuses incoming light rays and projects it onto the sensor grid. Modern

[2]Wei Sun and J.R. Cooperstock, "Requirements for Camera Calibration: Must Accuracy Come with a High Price?," in (Breckenridge, CO: IEEE, 2005), accessed January 16, 2024, `http://ieeexplore.ieee.org/document/4129503/`, `10.1109/ACVMOT.2005.102`.

cameras have compound lenses (lenses made up of several lens elements) in order to minimize undesired effects such as aberration, blurriness, and distortion.

- **Aperture** – Controls the amount of light that reaches the sensor. By adjusting the aperture size, the exposure and depth of field can be modified.
- **Sensor Grid** – Captures incoming light rays and converts this information into pixels on an image.

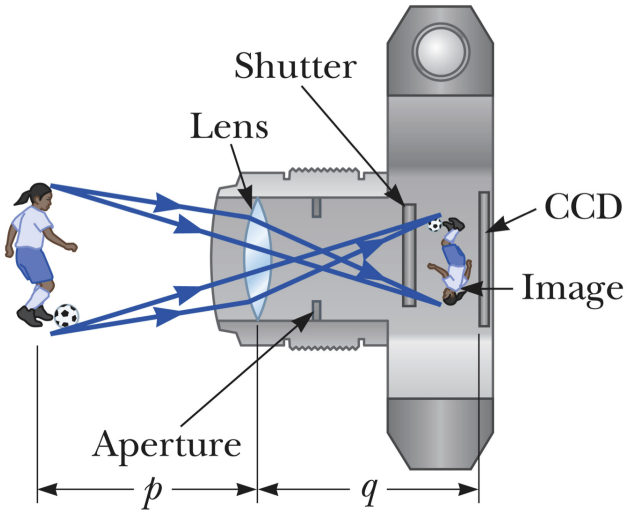


Figure 2.1: Lens camera. Adapted from John Colton, "Physics 123 Lecture 30 Warm-up Questions," BYU Physics and Astronomy, November 5, 2012, accessed October 17, 2023, `https://physics.byu.edu/faculty/colton/docs/phy123-fall12/jitt30a.html`.

However, it is impossible to construct a model which is both simple and exact for the lens cameras, as the behavior of lenses are very complex. As such, it is mathematically convenient to approximate the camera as a pinhole camera. In doing so, we ignore lens distortion, but it distills the behavior of a camera to its most fundamental and essential dynamics: the projection of points in 3D space onto the flat 2D image plane.

2.1.1 Pinhole Camera Model

A pinhole camera is a simple camera without a lens. It instead relies on the use of a tiny hole as the aperture of the camera, and light rays pass through the hole, projecting an inverted image onto the image plane. The pinhole camera model is based on the pinhole camera, however it goes further by making the assumption that the aperture is infinitely small. This

means that any incoming light ray would only travel in straight lines, going through the pinhole mapping to one singular point on the image plane.

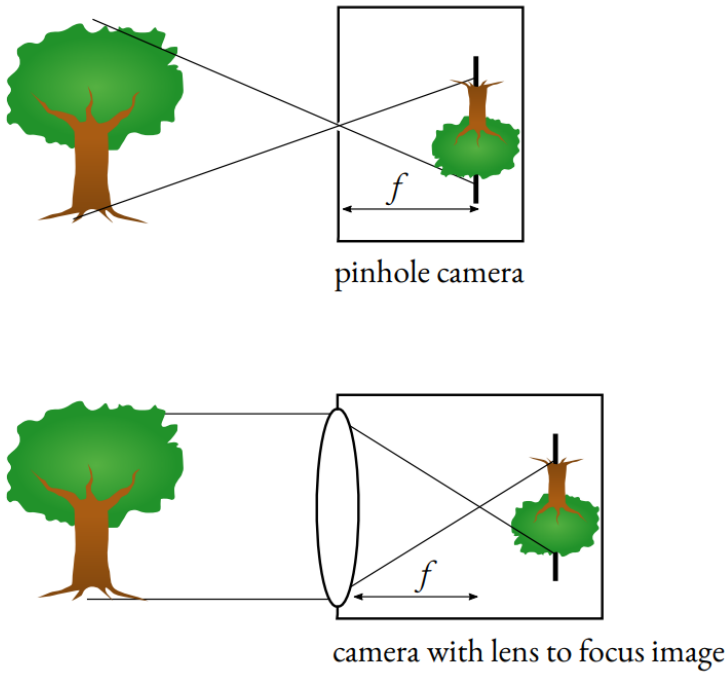


Figure 2.2: Difference between a pinhole camera and a lens camera. Adapted from Hoàng-Ân Lê, "Camera Model: Intrinsic Parameters," July 30, 2018, accessed October 14, 2023, `https://lhoangan.github.io/camera-params/`.

If necessary, one could reintroduce distortion and shear terms in order to minimize the error, but this is often not needed for low to medium precision applications, as the distortion of modern lenses are already minimal. The pinhole camera model is a sufficient, and its simplicity has led it to become one of the most frequently employed camera models in the field of camera calibration.

2.2 Calibration Object

The calibration object is an object with known dimensions and features which is often employed in camera calibration to establish a mapping between the 3D world and 2D image space. By capturing images of the calibration object, one can establish correspondences between points in the scene and pixels in the images, which can then be used to deduce the camera's intrinsic and extrinsic parameters. Calibration objects can be constructed in

many ways, and they can be separated into different categories based the dimension of the calibration object[3].

- **3D object based calibration** – Performed by using a calibration object whose geometry is known to very high precision. Typically, the calibration object consists of 2 or 3 orthogonal planes, although a plane whose precise translation is known may also be used, which also yields 3D reference points[4]. Using 3D objects is typically preferred, as it yields the highest accuracy[5], and the mathematics required is the simplest.

- **2D plane-based calibration** – The most common technique is known as *Zhang's method*, and it requires a planar object (often a checkerboard pattern), and various pictures of this plane are taken at different orientations[6]. Knowledge of the translation of the plane is not necessary. Due to its easier setup and good accuracy, it is the best choice in most situations. In fact, the most commonly used camera vision programming library, `OpenCV`, is geared towards this type of calibration.

- **1D line-based calibration** – Typically requires analyzing more than three photographs with straight lines which are not parallel with each other[7].

One can also calibrate cameras without a calibration object, using featuring tracking of objects in the scene to estimate camera parameters. This process is often referred to as self-calibration or auto-calibration. However, this is less preferable, as it involves a lot of estimation of parameters, which means that it becomes a more mathematically complex problem, and may not be able to achieve the level of accuracy of calibration achievable using known calibration patterns[8]. As such, it is typically the only chosen when pre-calibration is

[3]Zhengyou Zhang, "Camera Calibration," May 2007, accessed October 10, 2023, `https://people.cs.rutgers.edu/elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf`.

[4]Ibid.

[5]Ibid.

[6]Zhengyou Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, no. 11 (November 2000): 1330–1334, accessed October 1, 2023, `http://ieeexplore.ieee.org/document/888718/`, `10.1109/34.888718`.

[7]Xiuqin Chu, Fangming Hu, and Yushan Li, "Line-Based Camera Calibration," ed. Yue Hao et al., ed. David Hutchison et al., in *Computational Intelligence and Security*, vol. 3801 (Berlin, Heidelberg: Springer Berlin Heidelberg, 2005), accessed December 21, 2023, `http://link.springer.com/10.1007/11596448_150`, `10.1007/11596448_150`.

[8]Zhang, "Camera Calibration."

impossible.

For this paper, I will focus on 3D-based calibration, because the mathematics behind it is simpler. I think that this is sufficient because the complexity in other techniques typically lies in point extraction, which is hard in the 2D and 1D cases as it involves multiple images but is trivial for the 3D case. Once points are extracted, the same techniques can be then be utilized to generate the projection matrix. As such, concentrating on 3D-based calibration would allow us to gain a better understanding behind the general mathematical techniques and strategies employed in camera calibration.

3 Constructing the Pinhole Camera Model

At its core, the pinhole camera model is a mathematical model which describes how points in 3D space are projected onto a 2D plane. It defines the relationship between the 3D world, the pinhole camera, and the resulting 2D image.

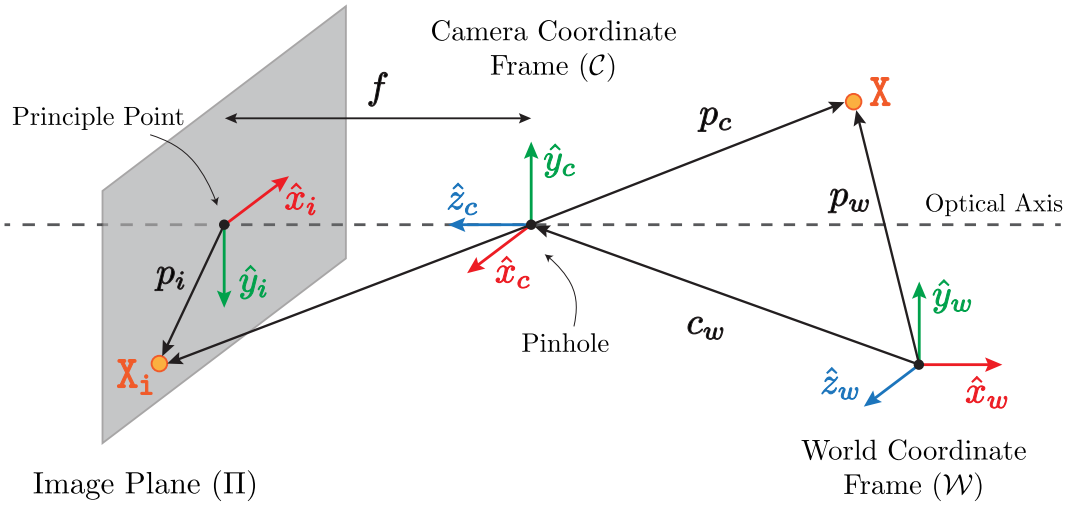


Figure 3.1: Pinhole camera model.

3.1 Nomenclature

- f – the focal length of the camera.

- $\boldsymbol{p_w}$ – the position vector of the point $\mathtt{X}$ in $\mathcal{W}$.
- $\boldsymbol{p_c}$ – the position vector of the point $\mathtt{X}$ in $\mathcal{C}$.
- $\boldsymbol{p_i}$ – the position vector of the projection $\mathtt{X_i}$ on the image plane relative to the principal point, which is where the optical axis intersects the image plane.
- $\boldsymbol{c_w}$ – the position vector of the pinhole in $\mathcal{W}$.

3.2 Coordinate Frames

To construct the pinhole camera model, we will need to define 4 different coordinate systems:

- **World Coordinate Frame ($\boldsymbol{\mathcal{W}}$)** – Represents the 3D space of the scene being photographed, with respect to an origin which may be arbitrary and depends on the conventions chosen. Objects that are in the scene are defined with respect to this coordinate frame.
- **Camera Coordinate Frame ($\boldsymbol{\mathcal{C}}$)** – Represents the 3D space of the scene with the pinhole (aperture) of the camera defined as the origin.
- **Image Coordinate Frame ($\boldsymbol{\Pi}$)** – 2D plane representing the image sensor plane of the camera. The origin is the principle point of the image sensor, where the optical axis intersects the image plane.
- **Pixel Coordinate Frame** – 2D plane representing the position of pixels on the image sensor. It is the discrete version of the image coordinate frame. The unit of measure is pixels.

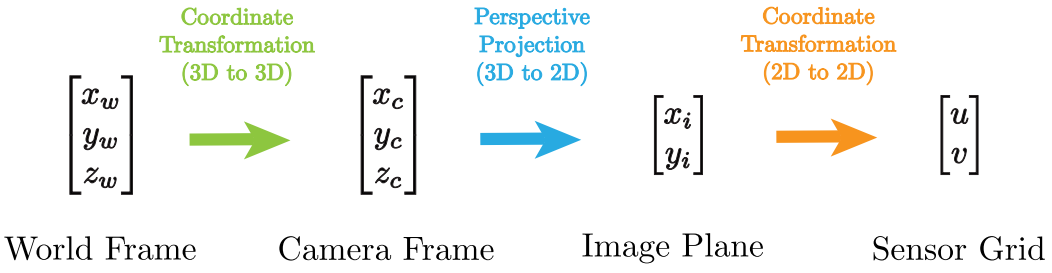


Figure 3.2: This workflow highlights the coordinate transformations that need to occur to project a point in the scene onto the image plane.

3.3 Intrinsic Parameters

Intrinsic parameters describe the internal characteristics of the camera. In other words, it dictates how in the 3D space are projected onto the image plane, i.e. the relationship between the position of `X` to its projection on the image plane.

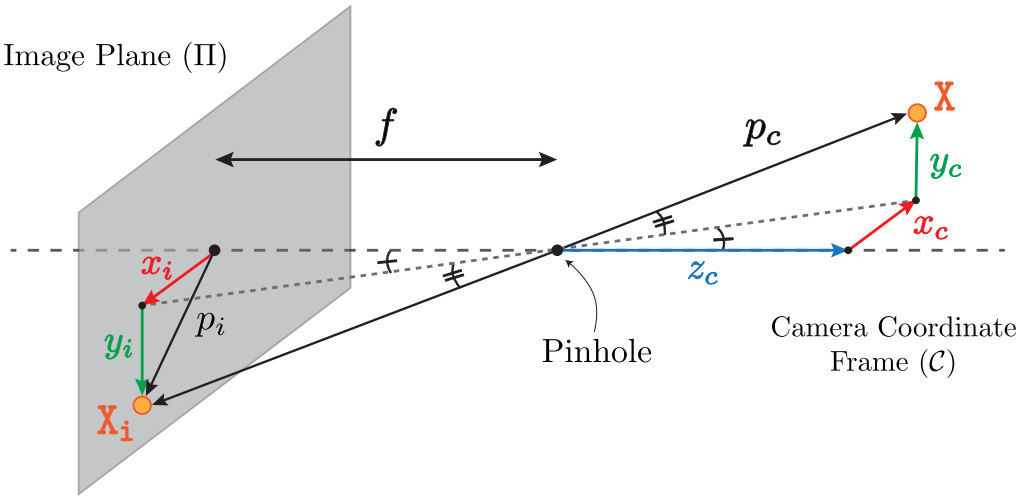


Figure 3.3: Perspective projection of the point `X` onto the image plane Π.

When a straight line is drawn from `X` to its projection $\mathtt{X_i}$ through the aperture, it intersects the optical axis. Deconstructing this intersection in the x and y direction, pairs of similar triangles are formed, which relates x_i to x_c and y_i to y_c.

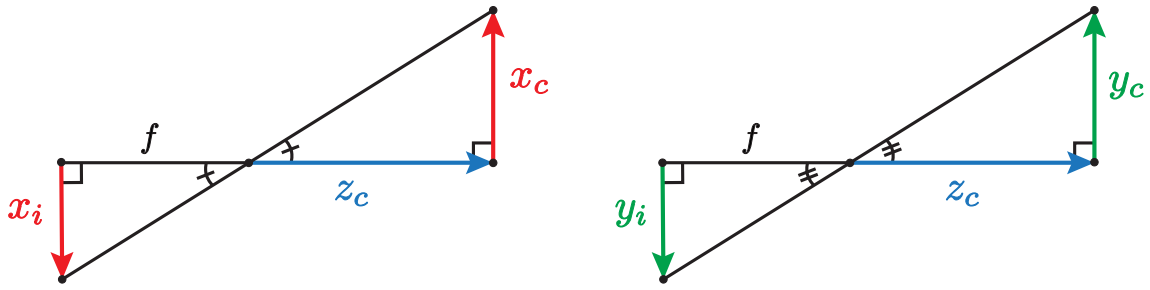


Figure 3.4: Similar triangles formed by perspective projection, which relate x_i to x_c and y_i to y_c.

$$\frac{x_i}{f} = \frac{x_c}{z_c} \quad \Longrightarrow \quad x_i = f\frac{x_c}{z_c} \tag{1a}$$

$$\frac{y_i}{f} = \frac{y_c}{z_c} \quad \Longrightarrow \quad y_i = f\frac{y_c}{z_c} \tag{1b}$$

Then, we need to convert the coordinates of the image projection $[x_i, y_i]^\mathsf{T}$ to actual pixel position of the point on the image $[u, v]^\mathsf{T}$. Pixel coordinates are measured in pixels, from the left-hand corner of the image. This is the convention that is typically followed in computer graphics. As such, there will be an offset in pixels, c_x and c_y, which represents the optical center of the image (i.e. the point at which the optical axis intersects the image plane). The relationship between $[x_i, y_i]^\mathsf{T}$ and $[u, v]^\mathsf{T}$ is proportional, but they scale at different rates, as $[x_i, y_i]^\mathsf{T}$ can be measured using any unit measurement, and can have negative and decimal values. On the other hand, $[u,v]^\mathsf{T}$ are measured in discrete pixel values, and since pixels are different sizes depending on the camera used, we define scaling factors, m_x and m_y, which represent the pixel density of the image sensor in the x and y axes of the image sensor plane respectively.

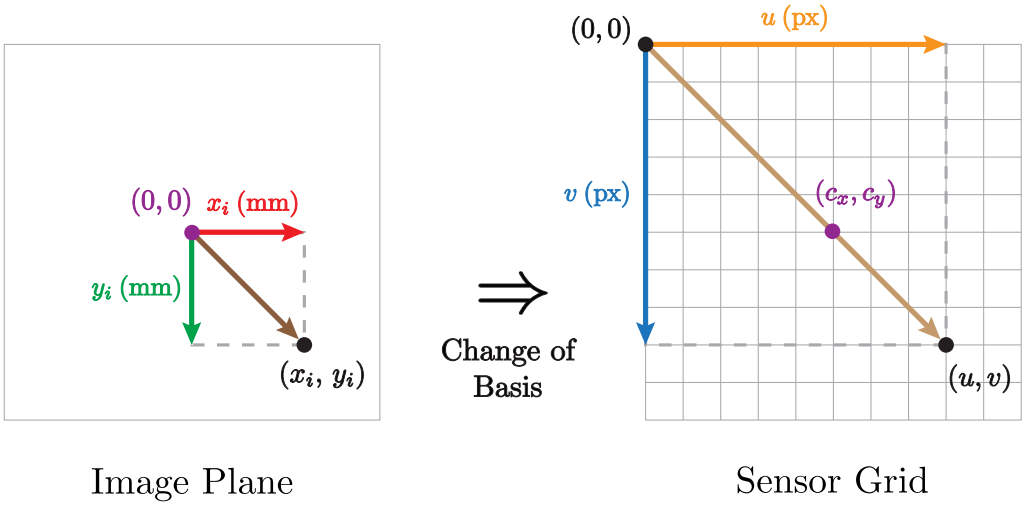


Figure 3.5: Conversion from image plane coordinates to sensor grid coordinates

Putting all the ideas above together, we can construct a set of linear parametric equations relating the pixel coordinates to their image coordinates thus:

$$u = m_x x_i + c_x$$

$$v = m_y y_i + c_y$$

where $u, v \in \mathbb{Z}_*^+$. Replacing x_i and y_i for the result we obtained from equations 1a and 1b, we get:

$$u = m_x f \frac{x_c}{z_c} + c_x$$

$$v = m_y f \frac{y_c}{z_c} + c_y$$

This gives us a direct relationship between camera coordinates and their corresponding pixel coordinates. Since m_x, m_y, and f are all unknowns, we can combine the products $m_x f$ and $m_y f$ into to f_x and f_y respectively. Under this new scheme, we define f_x and f_y as the

horizontal and vertical focal lengths of camera.

$$u = f_x \frac{x_c}{z_c} + c_x$$

$$v = f_y \frac{y_c}{z_c} + c_y$$

Multiply both sides of the equations by z_c.

$$z_c u = f_x x_c + z_c c_x$$

$$z_c v = f_y y_c + z_c c_y$$

Doing so allows us to express the relationship as a matrix transformation using **homogeneous coordinates**[9], by letting $\widetilde{w} = z_c$.

$$\begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = a \begin{bmatrix} f_x x_c + z_c c_x \\ f_y y_c + z_c c_y \\ z_c \end{bmatrix} = a \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}, \; a \in \mathbb{R}; a \neq 0 \tag{4}$$

The equation includes a non-zero scale factor a because a property of homogeneous coordinates is that multiplying the homogeneous coordinate of a point by a non-zero scalar still represents the same point. Equation 4 can be represented more simply thus:

$$\widetilde{p_i} = aK\, p_c\,, \; a \in \mathbb{R}; a \neq 0 \tag{5}$$

K is known as the **calibration matrix**. It is a matrix transformation which maps a point represented in the camera coordinate frame to the coordinates of their projection onto the sensor plane. An important property worth noting is that K is an *upper triangular matrix*. It is a special kind of square matrix where all of its non-zero entries are above the main diagonal. This is an important property which we will exploit when extracting the K from the projection matrix in section 5.1.

[9] See Appendix A.

3.4 Extrinsic Parameters

Extrinsic parameters describe the orientation of the camera by establishing the relationship between the position of a point in the world coordinate frame and its coordinates in camera coordinates.

There are two possible types of movement affecting the orientation of the camera: rotation and translation. The rotation of the camera can be described using a 3×3 square matrix:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{6}$$

where:

- **Row 1**: Unit vector representing $\hat{x}_c$ after rotation.
- **Row 2**: Unit vector representing $\hat{y}_c$ after rotation.
- **Row 3**: Unit vector representing $\hat{z}_c$ after rotation.

R is an orthonormal matrix, because the row and column vectors of R have to be orthogonal. Orthonormality is important because it ensures that the scale of vectors do not change (meaning that determinant of R has to be 1) and that the orthogonality between vectors are maintained.

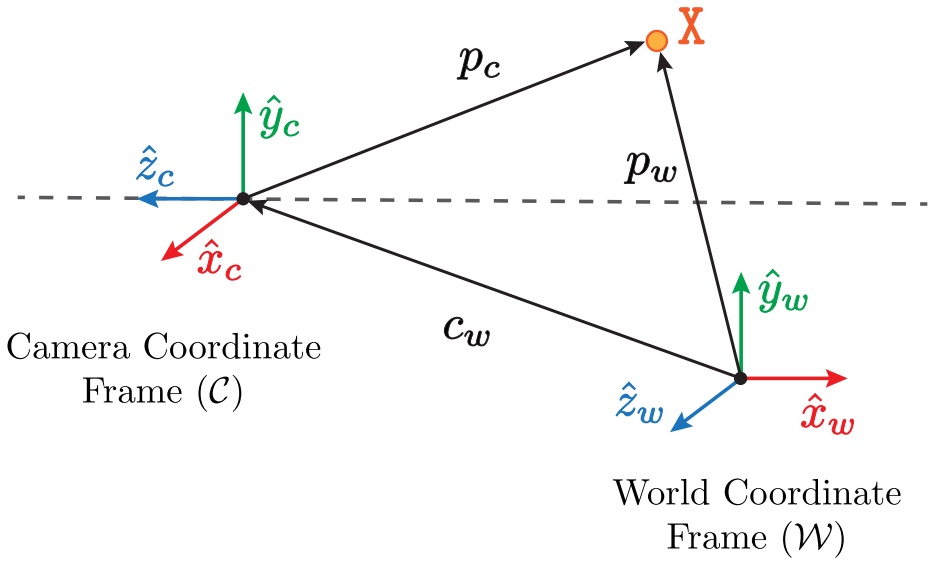


Figure 3.6: Coordinate transformation of `X` from the world coordinate frame to the camera coordinate frame.

From Figure 3.6, we can see that the naive approach to finding the position vector of the point in $\mathcal{C}$ is equal to the position vector of the point in $\mathcal{W}$ minus the position vector of the camera c_w. However, the camera can be facing in other directions, and we account for this rotation by including a rotational matrix in the equation. Thus:

$$\begin{aligned} p_c &= R\,(p_w - c_w) \\ &= R\,p_w - R\,c_w \end{aligned} \tag{7}$$

Since the position of the camera c_w is constant, we let $t = -R\,c_w$, where t represents the translation of the camera from the origin.

$$p_c = R\,p_w + t \tag{8}$$

This can be equivalently written as thus:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

We can combine R and t into an augmented matrix, $[R \,|\, t\,]$, by expressing p_w in homogeneous coordinates.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R \,|\, t\,]} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{9}$$

Thus, we arrive at the final equation:

$$p_c = [R \,|\, t\,]\, \widetilde{p}_w \tag{10}$$

4 Projection Matrix

When we combine the equation for the intrinsic transformation, $\widetilde{p}_i = aK\, p_c$ for $a \in \mathbb{R}$ and $a \neq 0$ (eq. 5), with the equation for the extrinsic transformation, $p_c = [R \,|\, t\,]\, \widetilde{p}_w$ (eq. 10), we obtain:

$$\widetilde{p}_i = aK\, [R \,|\, t\,]\, \widetilde{p}_w\,,\ a \in \mathbb{R}; a \neq 0 \tag{11}$$

This single equation encapsulates the relationship between the world coordinates and its corresponding pixel coordinates. We can then further simplify our camera model by defining a new matrix, P, which is equivalent to the product $K\, [R \,|\, t\,]$. Since K is a 3×3 matrix and

$[R\,|\,t]$ is a 3×4 matrix, the matrix product $K[R\,|\,t]$ yields a 3×4 matrix.

$$\underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{P} \equiv a \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R\,|\,t]} \tag{12}$$

Replacing P for $K\,[R\,|\,t]$ in equation 11, we obtain:

$$\widetilde{p}_i = aP\,\widetilde{p}_w \tag{13}$$

Equivalently:

$$\begin{bmatrix} \widetilde{u}_n \\ \widetilde{v}_n \\ \widetilde{w}_n \end{bmatrix} = a \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(n)} \\ y_w^{(n)} \\ z_w^{(n)} \\ 1 \end{bmatrix} \tag{14}$$

where $n \in \mathbb{N}$ denotes the n$^{\text{th}}$ point.

The implications of this equation is very important, as it means that a 3×4 matrix is sufficient in describing the relationship between a point in the world coordinate frame to its projection onto the image plane in pixel coordinates.

4.1 Solving for the Projection Matrix

Now, we need to devise a way to solve for the projection matrix. Since we know that the projection matrix describes relationship between a point and their projection, we can go backwards and solve for the projection matrix given a set of points and their corresponding image projections. Rewriting the matrix equation 13 as a set of parametric equations and

letting $a = 1$ for now, we obtain:

$$\widetilde{u}_n = p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14}$$

$$\widetilde{v}_n = p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24}$$

$$\widetilde{w}_n = p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}$$

We convert the set of equations back to their inhomogeneous form by recognizing the fact that $u_n = \widetilde{u}_n/\widetilde{w}_n$ and $v_n = \widetilde{v}_n/\widetilde{w}_n$.

$$u_n = \frac{p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14}}{p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}}$$

$$v_n = \frac{p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24}}{p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}}$$

For both equations, multiply both sides by the denominator.

$$u_n(p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}) = p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14}$$

$$v_n(p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}) = p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24}$$

Bringing all the terms onto one side:

$$0 = p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14} - p_{31}u_nx_w^{(n)} - p_{32}u_ny_w^{(n)} - p_{33}u_nz_w^{(n)} - p_{34}u_n \tag{15a}$$

$$0 = p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24} - p_{31}v_nx_w^{(n)} - p_{32}v_ny_w^{(n)} - p_{33}v_nz_w^{(n)} - p_{34}v_n \tag{15b}$$

The projection matrix P has 12 degrees of freedom, and since each point generates a distinct set of the two equations above, a minimum of 6 sets of points and their corresponding image projections are necessary in order to solve the system. These equations form a system

of equations, which can be rewritten in the form of a matrix equation:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1 x_w^{(1)} & -u_1 y_w^{(1)} & -u_1 z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1 x_w^{(1)} & -v_1 y_w^{(1)} & -v_1 z_w^{(1)} & -v_1 \\ & & \vdots & & & & & & & \vdots & & \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_n x_w^{(n)} & -u_n y_w^{(n)} & -u_n z_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_n x_w^{(n)} & -v_n y_w^{(n)} & -v_n z_w^{(n)} & -v_n \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}}_{p} \tag{16}$$

where G is a $2n \times 12$ matrix, $n \in \mathbb{Z}^+$, and p is the matrix vectorization of P. When using 6 correspondences, a unique solution can be obtained using classical approaches, such as using Gaussian elimination. But since we want to minimize uncertainty and achieve a solution which is as accurate as possible, we want to use as many correspondences as possible. When using more the 6 correspondences, we have more equations than unknowns. Such systems are *overdetermined*, and generally have no solutions[10]. However, we can optimize the system and obtain the "best approximate solution" using a method such as the **least-squares method**.

4.2 Constrained Least-Squares Solution

First, we must clarify what a "best approximate solution" is.

Definition 4.1 *Let A be an $m \times n$ matrix and let b be a vector in $\mathbb{R}^m$. A **least-squares solution** of the matrix equation $Ax = b$ is a vector $\widehat{x}$ such that:*

$$\operatorname{dist}(b, A\widehat{x}) \leq \operatorname{dist}(b, Ax) \qquad \forall x \in \mathbb{R}^m$$

[10]Gareth Williams, "Overdetermined Systems of Linear Equations," *The American Mathematical Monthly* 97, no. 6 (June 1990): 511–513, accessed November 1, 2023, https://www.jstor.org/stable/2323837, JSTOR: 2323837.

The term least-squares solution comes from the fact that $\text{dist}(b, A\widehat{x}) = \|b - A\widehat{x}\|$ is the square root of the sum of the squares of the entries of the vector $b - A\widehat{x}$. In other words, a least-squares solution solve a matrix equation as closely as possible by minimizing the sum of the squares of the differences between the entries of $A\widehat{x}$ and b.[11]

Returning to our specific problem, we can apply least-squares method by minimizing $\|Gp\|^2$. We minimize $\|Gp\|^2$ because we recognize that minimizing the square of the magnitude is equivalent to minimizing the magnitude itself, and that for a given vector v, it is easier to compute $\|v\|^2$ instead of $\|v\|$ since it eliminates the square root:

$$\|v\|^2 = \left(\sqrt{v_1^2 + v_2^2 + \cdots v_n^2}\right)^2 = v_1^2 + v_2^2 + \cdots v_n^2 = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = v^\mathsf{T} v$$

v^T represents the *transpose* of v, i.e. swapping its rows and columns, transforming a row vector into a column vector or vice versa.

However, we need to further constrain the problem since the projection matrix P is applied onto homogeneous coordinates, where the property holds that multiplying the homogeneous coordinate of a point by a non-zero scalar still represents the same point[12]. This means that there is an infinite amount of solutions, because once we have found a valid solution for P, we can always multiply P by a non-zero scalar k and still yield a valid solution, i.e. $P \equiv kP$. As such, any solution for P is only defined to a certain scale factor k. To mitigate this, we arbitrarily set the condition that $\|p\|^2$ must equal 1. Thus:

$$\underset{p}{\text{minimize}} \ \|Gp\|^2 \ \text{ subject to } \ \|p\|^2 = 1$$

Or equivalently:

$$\underset{p}{\text{minimize}} \ \left(p^\mathsf{T} G^\mathsf{T} G p\right) \ \text{ subject to } \ p^\mathsf{T} p = 1 \tag{17}$$

[11] Dan Margalit, Joseph Rabinoff, and Ben Williams, "7.5 The Method of Least Squares," in *Interactive Linear Algebra* , accessed December 25, 2023, `https://personal.math.ubc.ca/~tbjw/ila/ila.pdf`.

[12] See Appendix A.

The *Lagrangian*[13] of this equation is:

$$\mathcal{L}(p, \lambda) = p^\mathsf{T} G^\mathsf{T} G p - \lambda \left(p^\mathsf{T} p - 1 \right) \tag{18}$$

where $\lambda \in \mathbb{R}$ is the *Lagrange multiplier*. Since p is minimized when $\mathcal{L}$ is minimized, we want to find the absolute minimum of $\mathcal{L}$. As such, we need to locate the critical points of $\mathcal{L}$. To find these points, we want to look for values of p and λ where all partial derivatives (denoted using ∂) of the Lagrangian are zero.

$$\begin{aligned} \frac{\partial}{\partial p} \mathcal{L}(p, \lambda) &\stackrel{\text{set}}{=} 0 \\ \Rightarrow \frac{\partial}{\partial p} \left[p^\mathsf{T} G^\mathsf{T} G p - \lambda \left(p^\mathsf{T} p - 1 \right) \right] &= 0 \\ \Rightarrow 2 G^\mathsf{T} G p - 2 \lambda p &= 0 \\ \Rightarrow G^\mathsf{T} G p &= \lambda p \end{aligned} \tag{19}$$

This equation is in fact in the form of the eigenvalue problem for $G^\mathsf{T} G$. Potential solutions for p are eigenvectors that satisfy equation 19, with the scalar $\lambda \in \mathbb{R}$ as the eigenvalue. Since this is a minimization problem, the best approximate solution to p is the one which has the smallest λ.[14]

5 Extracting Parameters

Once we have solved for the projection for the projection matrix P, we can then decompose it and extract the intrinsic and extrinsic parameters. Recall that $P = K \left[R \mid t \right]$. Distributing

[13] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley, "Eigenvalue and Generalized Eigenvalue Problems: Tutorial," Comment: 8 pages, Tutorial paper. v2, v3: Added additional information, May 20, 2023, 2, accessed October 21, 2023, `http://arxiv.org/abs/1903.11240`, arXiv: `1903.11240 [cs, stat]`.

[14] Ibid.

K into $[\,R \mid t\,]$ yields:

$$\begin{aligned} P &= [\,KR \mid Kt\,] \\ &= [\,KR \mid -KRc_w\,] \\ &= [\,Q \mid -Qc_w\,] \qquad \text{with } Q = KR \end{aligned} \tag{20}$$

Analyzing the above equation reveals that Q corresponds to the first three columns of the projection matrix. Thus:

$$Q \equiv \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{R}$$

5.1 RQ Decomposition

Since K is in the form of an *upper right triangular matrix* and R is an *orthonormal matrix*, it is possible to decompose P and find unique solutions for K and R using a method called **RQ decomposition**. There are various algorithms that exist to do so, such as the *Graham-Schmidt Process* and the *Householder transformation* method, however, the minute details of how they work fall beyond the scope of this paper. Instead, we use `SciPy`, a *Python* library, to perform the decomposition.

5.2 Extracting the Camera Position

From equation 20, we see that $-Qc_w$ is equal to the last column of P. Thus:

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = -Qc_w$$

$$\Rightarrow \boxed{c_w = -Q^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}} \tag{21}$$

where Q^{-1} denotes the inverse of Q.

5.3 Extracting Orientation as Angles

When constructing the extrinsic matrix in section 3.4, we defined the rotation of the camera as a 3×3 matrix, where the rows represented the unit vectors $\hat{x}_c$, $\hat{y}_c$, and $\hat{z}_c$ after the rotation is performed. However, it is easier to interpret the rotations in terms of angles because angles provide a more intuitive and human-understandable representation of orientation. Specifically, we can employ **Tait-Bryan angles**, where the rotation is decomposed into three sequential rotations about each of the principle axes. These angles offer a straightforward way to understand how an object is rotated in terms of pitch, yaw, and roll. We employ *x-y-z* Tait-Bryan angles, a variant where the x rotation is performed first, then the y rotation, and finally the z rotation. This can be mathematically represented thus:

$$R \equiv R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{22}$$

where:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{23a}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & -\cos(\beta) \end{bmatrix} \tag{23b}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{23c}$$

We can then multiply the matrices together to obtain a single 3×3 matrix:

$$R = \begin{bmatrix} \cos(\beta)\,\cos(\gamma) & \sin(\alpha)\,\sin(\beta)\,\cos(\gamma) - \cos(\alpha)\,\sin(\gamma) & \cos(\alpha)\,\sin(\beta)\,\cos(\gamma) + \sin(\alpha)\,\cos(\gamma) \\ \cos(\beta)\,\sin(\gamma) & \sin(\alpha)\,\sin(\beta)\,\sin(\gamma) + \cos(\alpha)\,\cos(\gamma) & \cos(\alpha)\,\sin(\beta)\,\sin(\gamma) - \sin(\alpha)\,\cos(\gamma) \\ -\sin(\beta) & \sin(\alpha)\,\cos(\beta) & \cos(\alpha)\,\cos(\beta) \end{bmatrix} \tag{24}$$

Since we have already determined for the parameters of the rotation matrix, we can solve for the angles α, β, and γ by deriving formulas for them using equation 24:

$$\begin{aligned} r_{31} &= -\sin(\beta) \\ \Rightarrow &\boxed{\beta = \sin^{-1}(-r_{31})} \end{aligned} \tag{25}$$

$$\begin{aligned} r_{32} &= \sin(\alpha)\cos(\beta) \\ \Rightarrow \sin(\alpha) &= \frac{r_{32}}{\cos(\beta)} = \frac{r_{32}}{\cos(\sin^{-1}(-r_{31}))} \\ \Rightarrow &\boxed{\alpha = \sin^{-1}\left(\frac{r_{32}}{\sqrt{1 - r_{31}^2}}\right)} \end{aligned} \tag{26}$$

$$r_{21} = \cos(\beta)\sin(\gamma)$$

$$\Rightarrow \sin(\gamma) = \frac{r_{21}}{\cos(\beta)} = \frac{r_{21}}{\cos(\sin^{-1}(-r_{31}))}$$

$$\Rightarrow \boxed{\gamma = \sin^{-1}\left(\frac{r_{21}}{\sqrt{1 - r_{31}^2}}\right)} \tag{27}$$

6 Experimental Validation

To demonstrate the effectiveness of the model, I conducted tests by applying it to calibrate three different cameras: the Canon EOS D80, the iPhone X, and the Nikon D100. These cameras were chosen because they vary widely in terms of construction, image quality, and level of distortion. This diverse set of cameras provides a robust evaluation platform, allowing for an empirical assessment of the model's calibration performance, despite the small and statistically insignificant sample size.

To calibrate the camera, I first captured a picture of the calibration object of known dimensions using each of the cameras. In this case, the calibration object used consisted of 3 orthogonal planes with a checkerboard pattern imprinted onto each of the planes. The points at which the planes intersect was chosen to be the origin point of the world scene.

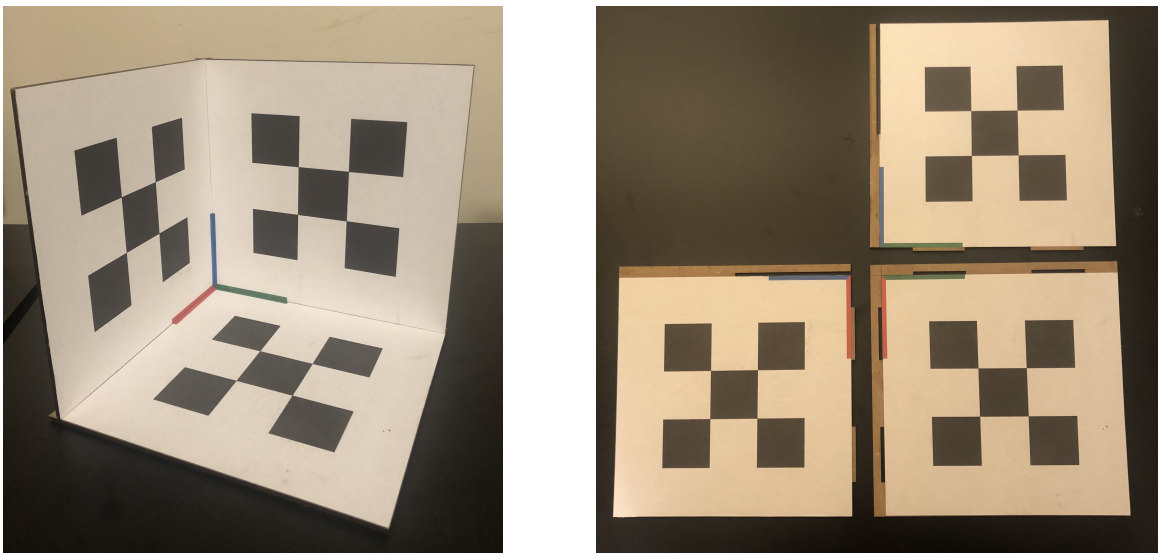


Figure 6.1: Custom-made calibration object made using laser-cut MDF board. The checkerboard pattern was printed onto paper and glued onto the boards.

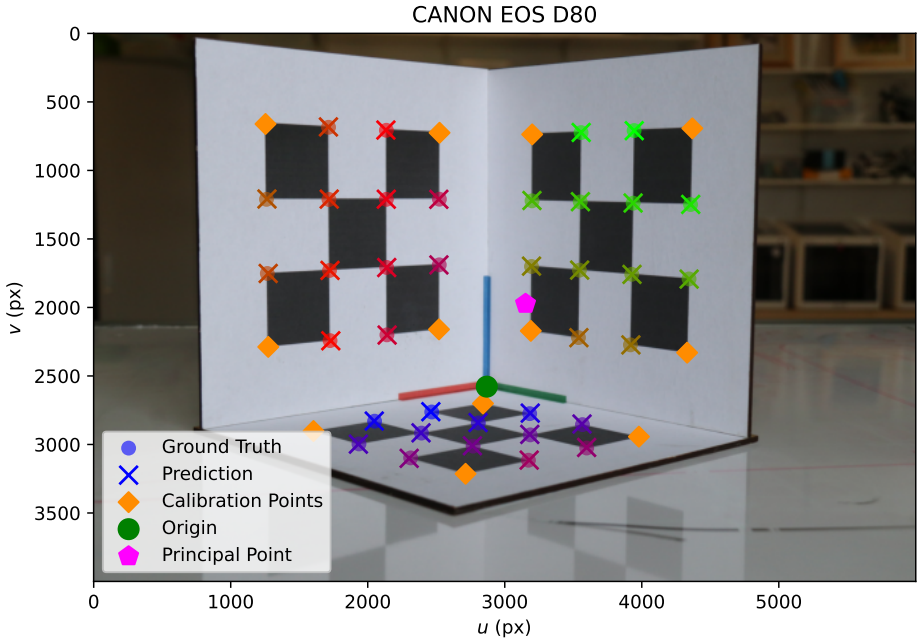
Then, I extracted the pixel coordinates of each of the points on the checkerboard and paired them to their corresponding real-world coordinates. This forms our dataset, which was then fed into a self-developed *Python* program named `calicam`[15]. A subset of the dataset was earmarked to calibrate the model, used to solve for the projection matrix and extract intrinsic and extrinsic parameters. The remaining points were utilized for validation; the program reprojects these points and compares the estimated pixel coordinates of the projections to the ground truth provided in the dataset. By taking the average distances between the predicted points and their ground truths, we essentially calculate the reprojection error of the model, providing a quantitative measure of the accuracy of the predicted image coordinates compared to the actual observed values.

		Canon EOS D80	**iPhone X**	**Nikon D100**
Focal Lengths	f_x	8404 px	3282 px	8144 px
	f_y	8388 px	3280 px	8143 px
Principal Point	c_x	3152 px	2043 px	1542 px
	c_y	1973 px	1453 px	1028 px
Tait-Bryan Angles	α	$-81.86°$	$-60.21°$	$-70.83°$
	β	$44.27°$	$38.72°$	$46.44°$
	γ	$4.97°$	$21.64°$	$13.89°$
Translation	t_x	494.8 mm	329.0 mm	840.3 mm
	t_y	537.6 mm	321.4 mm	766.0 mm
	t_z	128.3 mm	208.6 mm	317.2 mm
Reproj. Errors	μ_{max}	11.08 px	5.58 px	11.70 px
	μ_{avg}	3.56 px	2.55 px	2.81 px

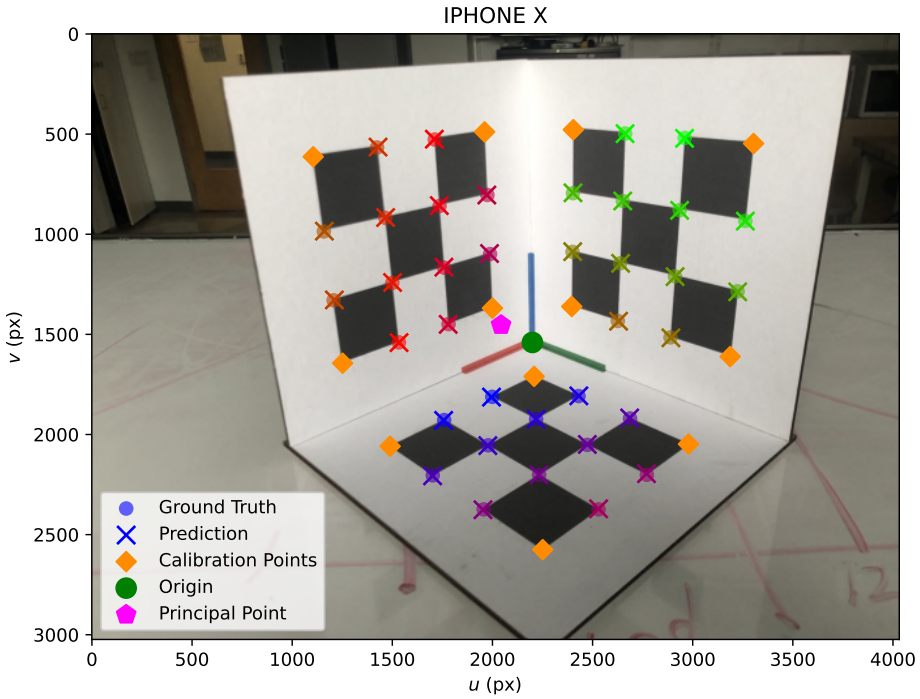
Table 6.1: Intrinsic and Extrinsic Parameters calculated by `calicam`.

[15]See Appendix C for the source code of `calicam`.

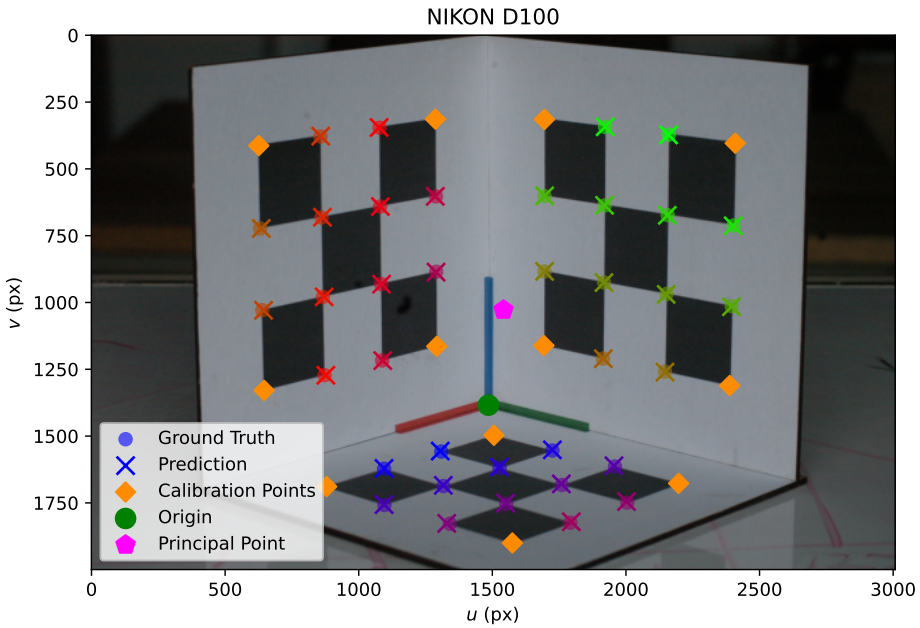
The program is also capable of creating graphs by overlaying the image with the predicted image coordinates based on inputs. This feature enables users to visualize and compare the position of the predicted coordinates with the actual observed ones, providing a visual representation which can be used to empirically observe the model's performance.



(a) Canon EOS D80.



(b) iPhone X.



(c) Nikon D100

Figure 6.2: Graphs generated by `calicam`.

6.1 Validating Estimated Focal Length

Given that specification of cameras are readily available online, we can actually further evaluate the accuracy of our model by calculating the focal lengths estimated by our model and comparing it to the manufacturer reported value. Assuming that the pixels are square, we estimate the focal lengths of our cameras to be the average of the horizontal and vertical focal lengths. Then, based on the manufacturer reported size of each individual pixel (known as the *pixel pitch*), we can convert our estimated focal length from pixels to millimeters.

	Calculated Focal Length[16]	**Reported Focal Length**	**% Error**
Canon EOS D80	$(8496\,\text{px})(3.73 \times 10^{-3}\,\text{mm/px}) \approx \boxed{31.7\,\text{mm}}$	32 mm	0.94 %
IPhone X	$(3280\,\text{px})(1.22 \times 10^{-3}\,\text{mm/px}) \approx \boxed{4.00\,\text{mm}}$	4 mm	–
Nikon D100	$(8143\,\text{px})(7.82 \times 10^{-3}\,\text{mm/px}) \approx \boxed{63.7\,\text{mm}}$	55 mm	15.6 %

Table 6.2: Comparison of Calculated vs. Manufacturer Reported Focal Length.

[16]Pixel pitches retrieved from `digicamdb.com`.

Considering that the focal lengths reported by manufacturers are often only accurate to around $\pm 1\,\text{mm}$,[17] my results are very promising, with exception to the Nikon D100. However, this error is in fact a result of human error, as I forgot to turn off autofocus on the Nikon D100, and the zoom lens on the Nikon D100 altered the effective focal length.

7 Conclusion

Based on my initial experimental validation, it is evident that the camera calibration method which I have devised displays a high level of accuracy. Despite the lack of consideration of lens distortion by model, the results still displayed negligible reprojection errors. The method I devised was simple, as it only required a single image containing the calibration object, however, there are various weaknesses to my method, most notably fixed focal length and limited adaptability to different localities. Despite this, it still serves as an excellent proof of concept which demonstrates the various strategies and mathematical techniques employed in camera calibration.

Acknowledgements

I am very grateful to my EE supervisor for his continual guidance and invaluable pieces of advice during the process of writing this extended essay. Additionally, I would like to express my appreciation to Mr. Auclair for dedicating his time to instruct me on camera operation and familiarizing me with camera settings. I am also indebted to Leon, for his help in operating the laser cutter. Last but not least, I want to thank Aditya for helping me troubleshoot issues that came up when using Adobe Illustrator for creating diagrams used in my paper.

[17] WayneF, "Answer to 'Are Lenses Marked with the True Focal Length?'," Photography Stack Exchange, August 7, 2017, accessed December 3, 2023, `https://photo.stackexchange.com/a/91603`.

Bibliography

Albertz, Joerg. "A Look Back; 140 Years of Photogrammetry." *Journal of the American Society for Photogrammetry and Remote Sensing* 73, no. 5 (May 2007): 504–506. Accessed September 9, 2023. https://www.asprs.org/wp-content/uploads/pers/2007journal/may/lookback.pdf.

Chu, Xiuqin, Fangming Hu, and Yushan Li. "Line-Based Camera Calibration." Edited by Yue Hao, Jiming Liu, Yuping Wang, Yiu-ming Cheung, Hujun Yin, Licheng Jiao, Jianfeng Ma, and Yong-Chang Jiao. Edited by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, et al. In *Computational Intelligence and Security*, 1009–1014. Vol. 3801. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. Accessed December 21, 2023. http://link.springer.com/10.1007/11596448_150. 10.1007/11596448_150.

Colton, John. "Physics 123 Lecture 30 Warm-up Questions." BYU Physics and Astronomy, November 5, 2012. Accessed October 17, 2023. https://physics.byu.edu/faculty/colton/docs/phy123-fall12/jitt30a.html.

Ghojogh, Benyamin, Fakhri Karray, and Mark Crowley. "Eigenvalue and Generalized Eigenvalue Problems: Tutorial." Comment: 8 pages, Tutorial paper. v2, v3: Added additional information. May 20, 2023. Accessed October 21, 2023. http://arxiv.org/abs/1903.11240. arXiv: 1903.11240 [cs, stat].

Lê, Hoàng-Ân. "Camera Model: Intrinsic Parameters." July 30, 2018. Accessed October 14, 2023. https://lhoangan.github.io/camera-params/.

Margalit, Dan, Joseph Rabinoff, and Ben Williams. "7.5 The Method of Least Squares." In *Interactive Linear Algebra*, 411–432. Accessed December 25, 2023. https://personal.math.ubc.ca/~tbjw/ila/ila.pdf.

WayneF. "Answer to 'Are Lenses Marked with the True Focal Length?'" Photography Stack Exchange, August 7, 2017. Accessed December 3, 2023. https://photo.stackexchange.com/a/91603.

Wei Sun and J.R. Cooperstock. "Requirements for Camera Calibration: Must Accuracy Come with a High Price?" In, 356–361. Breckenridge, CO: IEEE, 2005. Accessed January 16, 2024. http://ieeexplore.ieee.org/document/4129503/. 10.1109/ACVMOT.2005.102.

Williams, Gareth. "Overdetermined Systems of Linear Equations." *The American Mathematical Monthly* 97, no. 6 (June 1990): 511–513. Accessed November 1, 2023. https://www.jstor.org/stable/2323837. JSTOR: 2323837.

Zhang, Zhengyou. "A Flexible New Technique for Camera Calibration." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, no. 11 (November 2000): 1330–1334. Accessed October 1, 2023. http://ieeexplore.ieee.org/document/888718/. 10.1109/34.888718.

———. "Camera Calibration," May 2007. Accessed October 10, 2023. https://people.cs.rutgers.edu/elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf.

Appendix A Homogeneous Coordinates

Definition A.1 *Given a point* $[a_1, a_2, \cdots, a_n]^\mathsf{T} \in \mathbb{R}^n$*, it can be expressed in homogeneous coordinates as* $[\lambda a_1, \lambda a_2, \cdots, \lambda a_n, \lambda]^\mathsf{T} \in \mathbb{R}^{n+1}$*, where* $\lambda \neq 0$ *is a constant scale factor.*

In other words, homogeneous coordinates are a mathematical representation that extends the Cartesian coordinate system by introducing an additional coordinate. Consider the point $[u, v]^\mathsf{T}$ in 2D Euclidean space. We can convert it to homogeneous coordinates by adding a new coordinate, represented by $\widetilde{w}$:

$$\begin{bmatrix} u \\ v \end{bmatrix} \sim \begin{bmatrix} u\widetilde{w} \\ v\widetilde{w} \\ \widetilde{w} \end{bmatrix} \equiv \begin{bmatrix} \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \end{bmatrix} \tag{28}$$

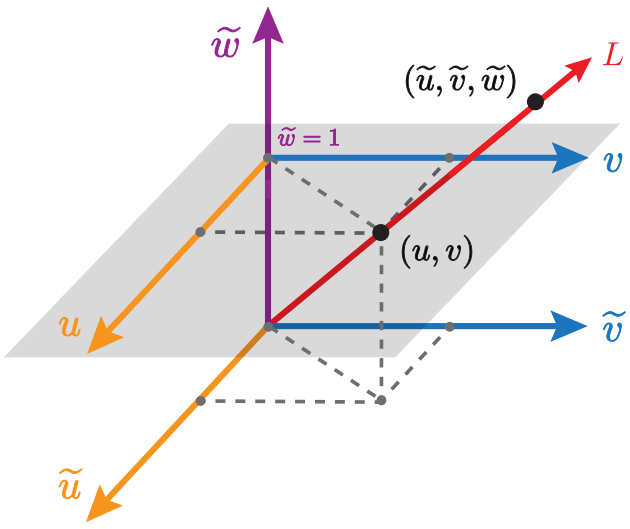


Figure A.1: Homogeneous coordinate system.

One important property of homogeneous coordinates is that if the homogeneous coordinate of a point is multiplied by a non-zero scalar, the result represents the same point, as visualized

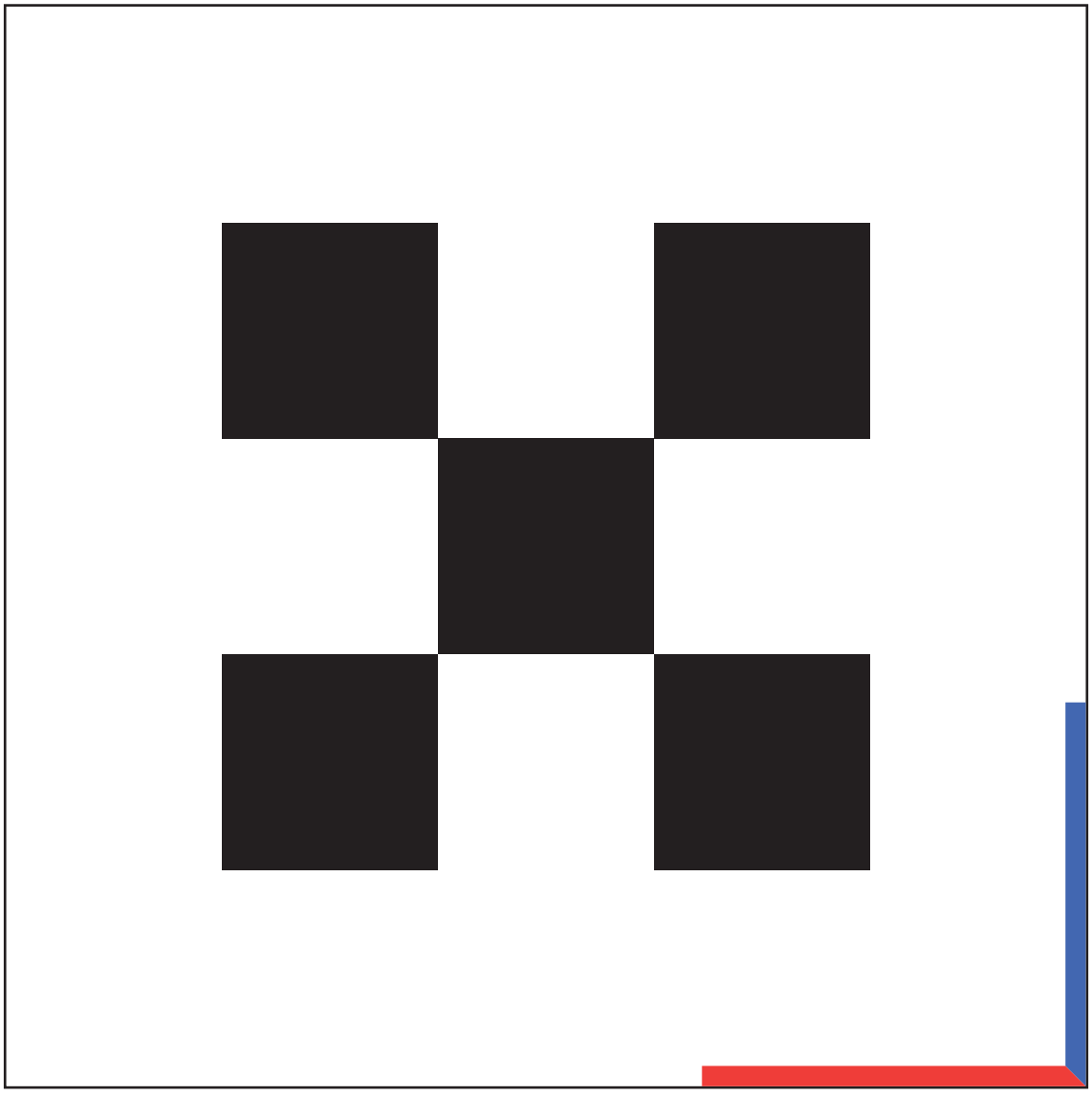
in Figure A.1. For example, given the homogeneous point $[\widetilde{u},\widetilde{v},\widetilde{w}]^\mathsf{T}$:

$$\begin{bmatrix} \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \end{bmatrix} \equiv k \begin{bmatrix} \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \end{bmatrix} \ , \ k \neq 0 \tag{29}$$

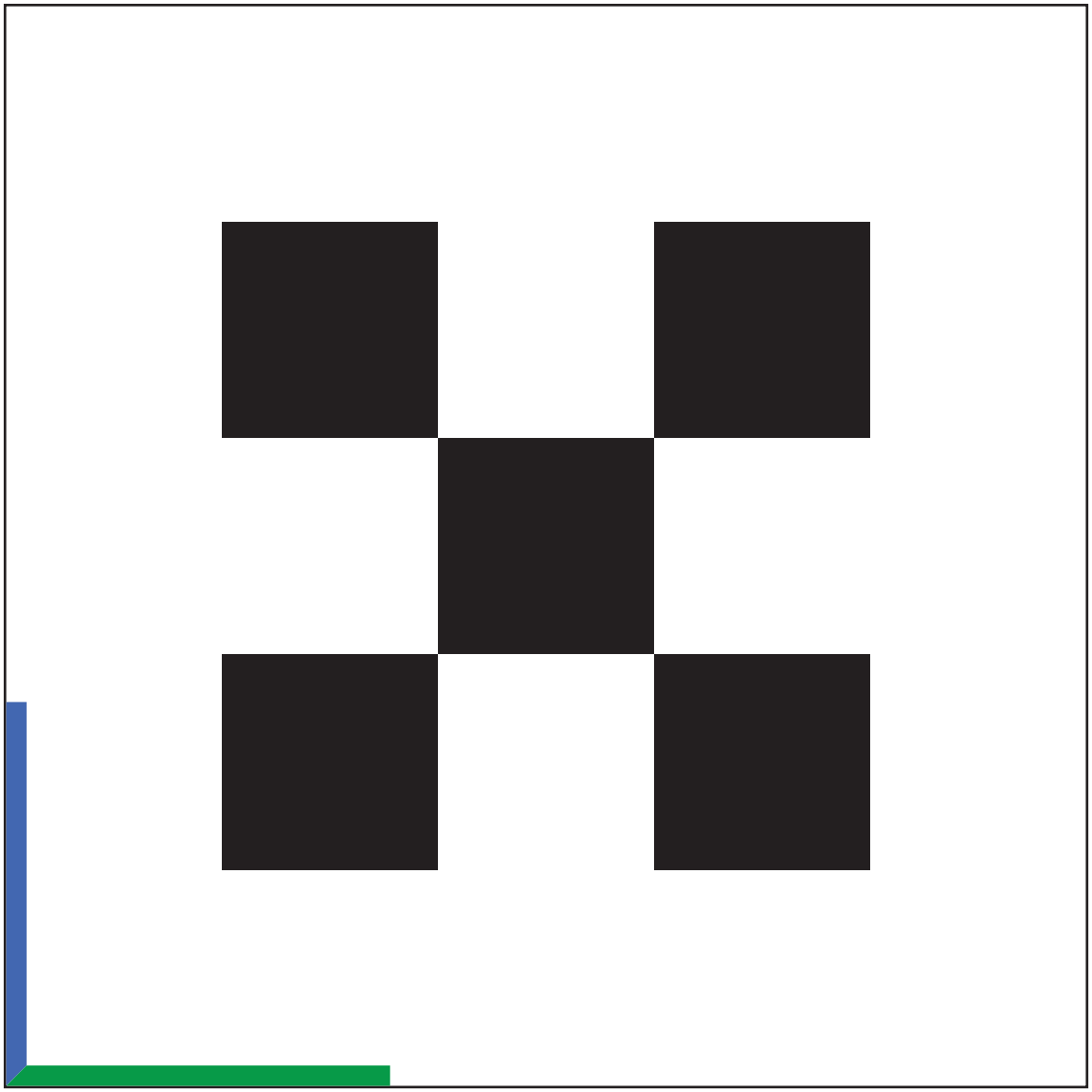
For this reason, homogeneous coordinates are able to simplify and generalize transformations, like translation, rotation, scaling, and perspective projection without the need for separate and complex formulas for each operation.

Appendix B Calibration Object Panel Patterns

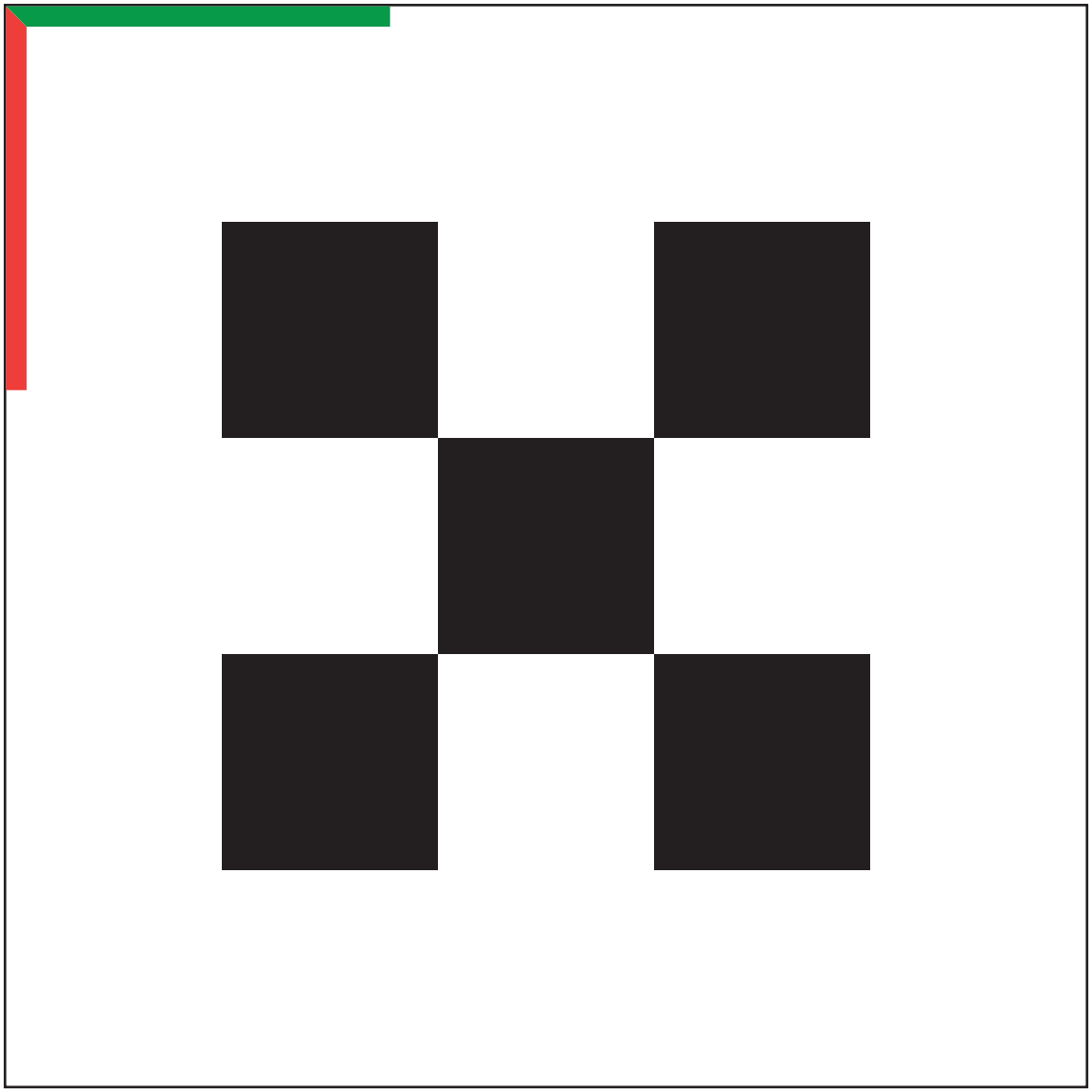
XZ Face



YZ Face



XY Face



Appendix C Source Code

Structure

```
calicam
├── calicam
│   ├── __init__.py
│   ├── extract.py
│   ├── parser.py
│   ├── projection.py
│   └── vecs.py
└── run.py
```

Entry Point

run.py

```python
#!/usr/bin/env python3
import os
import sys
import numpy as np
import matplotlib.pyplot as plt
from argparse import ArgumentParser, RawDescriptionHelpFormatter

import calicam


def main():
    np.set_printoptions(precision=3, suppress=True)

    MAX_HELP_POSITION = 40

    parser = ArgumentParser(
        prog="calicam",
        description=(
            "Generates projection matrix and calculates intrinsic and extrinsic parameters.\n"
            "CSV inputs are in the format: x,y,z,u,v where 3D point = (x, y, z) and 2D point = (u,v)"),
        formatter_class=lambda prog: RawDescriptionHelpFormatter(prog, max_help_position=MAX_HELP_POSITION),
    )

    parser.add_argument("path", metavar="PATH", help="path to csv file with calibration points")
    parser.add_argument(
        "-d", "--data", metavar="DATA_PATH", help="path to csv file with model verification data")
    parser.add_argument(
        "-g", "--graph", nargs="?", const="", metavar="BKGD_IMG", help="generate graph")
    parser.add_argument("-t", "--title", metavar="TITLE", help="title of graph (ignored if `-g` is not
    ↪  passed)")
    parser.add_argument("-s", "--show", action="store_true", help="show graph (only necessary if `-o` is
    ↪  passed)")
    parser.add_argument("-o", "--out", metavar="GRAPH_PATH", help="graph output location ")
    parser.add_argument("--noprint", action="store_true", help="don't print output to terminal")

    args = parser.parse_args()

    output = []

    try:
        # GENERATE MODEL
        csv_path: str = args.path
```

```
41
42          cali_world_coords, cali_image_coords = calicam.parse_data_from_csv(csv_path)
43          proj_matrix, cali_matrix, rot_matrix, (tx, ty, tz) = calicam.calibrate_camera(
44              cali_world_coords, cali_image_coords)
45
46          # origin
47          (ox, oy) = calicam.project_point(proj_matrix, (0.0, 0.0, 0.0))
48
49          # principal point, focal lengths
50          (cx, cy), (fx, fy) = calicam.extract_intrinsics(cali_matrix)
51
52          # tait-bryan angles
53          a, b, g = calicam.extract_orientation_zyx(rot_matrix)
54
55          output.append("\n" + "\n\n".join((
56              f"Projection Matrix: \n{proj_matrix}",
57              f"Calibration Matrix: \n{cali_matrix}",
58              f"Rotation Matrix: \n{rot_matrix}",
59              f"Focal Lengths: \n\tf_x = {fx:.2f} px \n\tf_y = {fy:.2f} px",
60              f"Principal Point: \n\tc_x = {cx:.2f} px \n\tc_y = {cy:.2f} px",
61              f"Translation: \n\tt_x = {tx:.2f} \n\tt_y = {ty:.2f} \n\tt_z = {tz:.2f}",
62              f"Orientation: \n\t\u03B1 = {a:.2f}° \n\t\u03B2 = {b:.2f}° \n\t\u03B3 = {g:.2f}°",
63          )))
64
65          # MODEL VALIDATION
66          data_path: str | None = args.data
67
68          if data_path is not None:
69              assert os.path.isfile(data_path), f"{data_path} does not exist."
70              assert data_path.endswith(".csv"), f'{data_path} does not end with the extension ".csv".'
71
72              data_world_coords, data_image_coords = calicam.parse_data_from_csv(data_path)
73
74              predicted_coords = [
75                  calicam.project_point(proj_matrix, world_coord) for world_coord in data_world_coords
76              ]
77              reproj_errs = [
78                  calicam.euclidean(actual_coord, reproj_coord)
79                  for actual_coord, reproj_coord in zip(data_image_coords, predicted_coords)
80              ]
81
82              max_err = max(reproj_errs)
83              avg_err = sum(reproj_errs) / len(reproj_errs)
84
85              output.append(
86                  f"\nReprojection Errors: \n\t\u03BC_max = {max_err:.3f} px \n\t\u03BC_avg = {avg_err:.3f}
                    ↪  px")
87
88          # OUTPUT
89          if not args.noprint:
90              print(*output, sep="\n")
91
92          # GRAPH
93          image_path: str | None = args.graph
94
95          if image_path is not None:
96              ax: plt.Axes
97              _, ax = plt.subplots(figsize=(8, 10))
98
99              plt.gca().invert_yaxis()
100
101             # image was provided
102             if image_path != "":
103                 assert os.path.isfile(image_path), f"{image_path} does not exist."
104                 assert args.data, "Path to data csv file must be provide using -d flag to produce graph."
105
106                 img = plt.imread(image_path,)
107                 ax.imshow(img, cmap='gray')
```

```
108                 ax.autoscale(False)
109
110             # graph data points and model points only if -d flag is specfied
111             if data_path is not None:
112                 cmap = plt.cm.brg
113                 discrete_cmap = list(cmap(np.linspace(0, 1, len(data_image_coords))))
114
115                 # data points
116                 ax.scatter(
117                     *zip(*data_image_coords),
118                     label="Ground Truth",
119                     s=50,
120                     color=discrete_cmap,
121                     marker="o",
122                     alpha=0.6,
123                 )
124
125                 # predicted points
126                 ax.scatter(
127                     *zip(*predicted_coords),
128                     label="Prediction",
129                     s=100,
130                     color=discrete_cmap,
131                     marker="x",
132                 )
133
134             # calibration points
135             ax.scatter(
136                 *zip(*cali_image_coords),
137                 label="Calibration Points",
138                 s=60,
139                 marker="D",
140                 color="darkorange",
141             )
142
143             # origin point
144             ax.scatter(ox, oy, label="Origin", s=120, marker="o", color="green")
145
146             # principle point
147             ax.scatter(cx, cy, label="Principal Point", s=120, marker="p", color="magenta")
148
149             graph_title: str = args.title or image_path
150
151             plt.gca().update({"title": graph_title, "xlabel": "$u$ (px)", "ylabel": "$v$ (px)"})
152             plt.legend()
153
154             out_path: str | None = args.out
155
156             if out_path:
157                 plt.savefig(out_path, bbox_inches='tight')
158
159             # show graph if -s flag was specified or if a save location was not specified
160             if args.show or not out_path:
161                 plt.show()
162
163     except AssertionError as e:
164         parser.error(str(e))  # pass error to argparse
165
166     except KeyboardInterrupt:
167         print(f"\nKeyboardInterrupt")
168         sys.exit(1)
169
170     sys.exit(0)
171
172
173 if __name__ == "__main__":
174     main()
```

`calicam` Internal Library

calicam/parser.py

```python
import csv

from .vecs import *


def parse_data_from_csv(path: str) -> tuple[list[Vec3f], list[Vec2f]]:
    """
    Parses a csv and returns a list of 3D scene points and their corresonding
    2D image mappings.
    CSV format: x,y,z,u,v
    where 3D point = (x, y, z) and 2D point (u,v)
    """
    with open(path, "r") as f:
        reader = csv.reader(f, delimiter=",")

        world_coords = []
        image_coords = []
        for lno, line in enumerate(reader, start=1):
            assert len(line) == 5, f"Data on line {lno} in {path} is invalid."

            x, y, z, u, v = (float(s) for s in line)

            world_coords.append((x, y, z))
            image_coords.append((u, v))

    return world_coords, image_coords
```

calicam/projection.py

```python
import numpy as np
import scipy.sparse.linalg
from nptyping import Shape, Double

from .vecs import *

ProjMatrix = np.ndarray[Shape["3, 4"], Double]


def generate_estimation_matrix(world_coords: list[Vec3f], image_coords: list[Vec2f]) -> np.ndarray:
    """
    Generates an estimation matrix from list of 3D world coords and
    their corresponding pixel coord mappings
    """
    rows = []
    for (x, y, z), (u, v) in zip(world_coords, image_coords):
        rows.append([x, y, z, 1.0, 0.0, 0.0, 0.0, 0.0, -u * x, -u * y, -u * z, -u])
        rows.append([0.0, 0.0, 0.0, 0.0, x, y, z, 1.0, -v * x, -v * y, -v * z, -v])
    return np.array(rows)


def generate_proj_matrix(world_coords: list[Vec3f], image_coords: list[Vec2f]) -> tuple[ProjMatrix, float]:
    """
    Takes 3D calibration points their corresponding pixel coord mappings and
    returns the projection matrix as a 3x4 matrix
    """
    assert len(world_coords) == len(image_coords), \
        f"The number of world coordinates ({world_coords}) and image coordinates ({image_coords}) do not
        ↪  match."
```

```

    assert len(world_coords) >= 6, \
        f"Need at least 6 calibration points, but only {len(world_coords)} were provided."

    G = generate_estimation_matrix(world_coords, image_coords)
    M = G.T @ G

    eigval, p = scipy.sparse.linalg.eigs(M, k=1, which="SM")  # solve for minimum p using eigenvalue problem
    proj_matrix = p.real.reshape(3, 4)  # take only real part of p and convert into 3x4 matrix

    return proj_matrix, eigval


def project_point(projection_matrix: ProjMatrix, world_coords: Vec3f) -> Vec2f:
    """
    Calculate pixel coordinate from 3D world coord using projection matrix
    """
    return to_inhomogenous(projection_matrix @ to_homogenous(world_coords))  # turn into inhomogenous coords
```

calicam/extract.py

```
import numpy as np
import scipy.linalg
from math import sqrt
from nptyping import  Shape, Double

from .projection import generate_proj_matrix, ProjMatrix
from .vecs import *

CalMatrix = np.ndarray[Shape["3, 3"], Double]
RotMatrix = np.ndarray[Shape["3, 3"], Double]


def calibrate_camera(world_coords: list[Vec3f],
                     image_coords: list[Vec2f]) -> tuple[ProjMatrix, CalMatrix, RotMatrix, Vec3f]:
    """
    Decomposes the projection matrix into the calibration matrix,
    rotation matrix, and translation matrix.
    """
    proj_matrix, _ = generate_proj_matrix(world_coords, image_coords)

    K, R = scipy.linalg.rq(proj_matrix[:, :3])  # rq decomposition

    # enforce positive diagonal on K
    D = np.diag(np.sign(np.diag(K)))
    K = K @ D
    R = D @ R

    # scale projection matrix and calibration matrix to reflect real world scaling
    scale_factor = 1 / K[2][2]
    proj_matrix *= scale_factor
    K *= scale_factor

    # extract translation vector from P
    t = tuple(-np.linalg.inv(proj_matrix[:, :3]) @ proj_matrix[:, 3])

    return proj_matrix, K, R, t


def extract_intrinsics(K: CalMatrix) -> tuple[Vec2f, Vec2f]:
    """
    Extract principle point and focal lengths from calibration matrix
    """
```

```
    principal_point = (K[0][2], K[1][2])
    focal_lengths = (K[0][0], K[1][1])
    return principal_point, focal_lengths


def extract_orientation_zyx(R: RotMatrix) -> Vec3f:
    """
    Extract tait-bryan angles (zyx) from rotation matrix
    """
    return (
        np.degrees(np.arcsin(R[2][1] / sqrt(1 - (R[2][0])**2))),  # alpha (x rotation)
        np.degrees(np.arcsin(-R[2][0])),  # beta (y rotation)
        np.degrees(np.arcsin(R[1][0] / sqrt(1 - (R[2][0])**2))),  # gamma (z rotation)
    )
```

calicam/vecs.py

```
from math import sqrt

Vecf = tuple[float, ...]

Vec2f = tuple[float, float]
Vec3f = tuple[float, float, float]


def to_homogenous(vec: Vecf) -> Vecf:
    return (*vec, 1.0)


def to_inhomogenous(vec: Vecf) -> Vecf:
    return tuple(map(lambda v_i: v_i / vec[-1], vec[:-1]))


def euclidean(a: Vecf, b: Vecf) -> float:
    assert len(a) == len(b), f"Vectors need to have the same dimension"
    return sqrt(sum((b_i - a_i)**2 for a_i, b_i in zip(a, b)))
```