IB Math Analysis and Approaches (HL)

Extended Essay

May 2024 Session

The Mathematics Techniques of Camera Calibration

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

Word Count: 3066 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 mathematical techniques use in camera calibration, and why they find applications across many real-world applications.

2 Approach

There are many techniques one could take to calibrate a camera,

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 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.

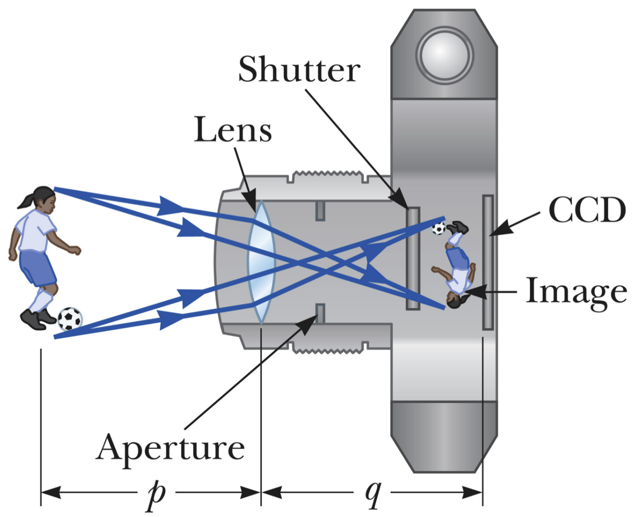


Figure 2.1: Lens camera. Adapted from John Colton, "Physics 123 Lecture 30 Warm-up Questions," BYU Physics and Astronomy, November 5, 2012, accessed October 17, 2023, `https://physics.byu.edu/faculty/colton/docs/phy123-fall12/jitt30a.html`.

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.

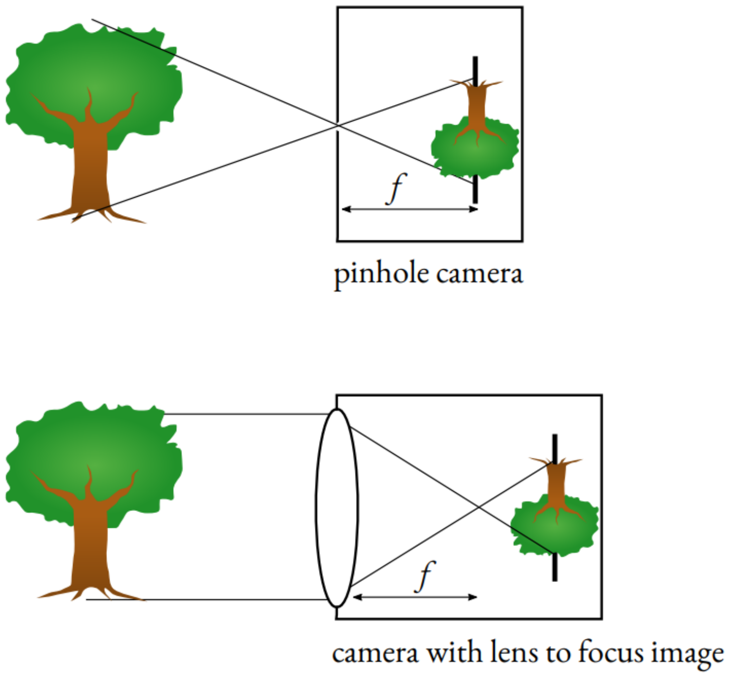


Figure 2.2: Difference between a pinhole camera and a lens camera. Adapted from Hoàng-Ân Lê, "Camera Model: Intrinsic Parameters," July 30, 2018, accessed October 14, 2023, `https://lhoangan.github.io/camera-params/`.

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. As such, its ease of use 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 used in camera calibration to

Calibration objects can be constructed in many ways, and they can be separated into different categories based the dimension of the calibration object[2].

- **3D object based calibration** – Performed by using a calibration object whose geometry is known to very high precision. Typically, the calibration object consists

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

of 2 or 3 orthogonal planes, although a plane whose precise translation is known may also be used, which also yields 3D reference points[3]. Using 3D objects is typically preferred, as it yields the highest accuracy[4], and the mathematics required is also the least.

- **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[5]. 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[6].

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 not only means that it is a more mathematically complex problem, it may not be able to achieve the accuracy of calibration using known calibration patterns[7]. As such, it is typically the only chosen when pre-calibration is impossible.

For this paper, I will focus on calibration using a 3D calibration object, because the mathematics behind it is simpler, and many of the techniques used in 3D-based calibration are

[3]Zhang, "Camera Calibration."

[4]Ibid.

[5]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`.

[6]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`.

[7]Zhang, "Camera Calibration."

3 Prerequisites

3.1 Notation

Vectors and Matrices. In this paper, lowercase letters are used to denote vectors, whereas capital letters are used for matrices.

Transpose of Vectors and Matrices. The transpose of a vector or a matrix is an operation whereby the rows and columns of the vector or matrix are inverted, and this is denoted using the notation v^T or M^T. For example:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^\mathsf{T} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

3.2 Homogenous Coordinates

While Euclidean space describes 2D and 3D space well, they are not sufficient in describing perspective projections, as it is unable to fully the capture the relationships inherent in projective projections and affine transformations, both of which are core concepts in this paper.

Homogenous coordinates forms the basis of projective geometry, because it unifies the treatment of common graphical transformations such as rotation and translations[8].

Given a point in with coordinates $(a_1, a_2, \cdots, a_n) \in \mathbb{R}^n$

Given the vector $[\,u, v\,]^\mathsf{T} \in \mathbb{R}^2$, we can express it in terms of homogenous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} u\widetilde{w} \\ v\widetilde{w} \\ \widetilde{w} \end{bmatrix} \equiv \begin{bmatrix} \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \end{bmatrix} \tag{3.1}$$

[8]Jules Bloomenthal and Jon Rokne, "Homogeneous Coordinates," *The Visual Computer* 11, no. 1 (January 1994): 1, accessed October 20, 2023, `http://link.springer.com/10.1007/BF01900696`, `10.1007/BF01900696`.

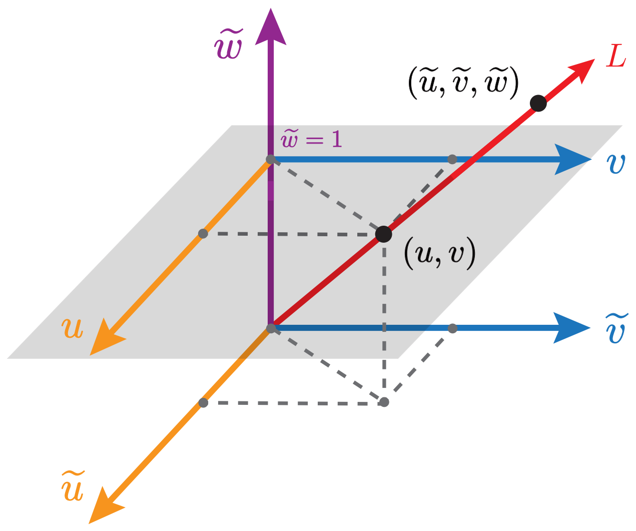


Figure 3.1: Homogenous coordinate system.

In other words, with homogenous coordinates, we interpret our *Euclidean* space as an *affine* space

4 Constructing the Pinhole Camera Model

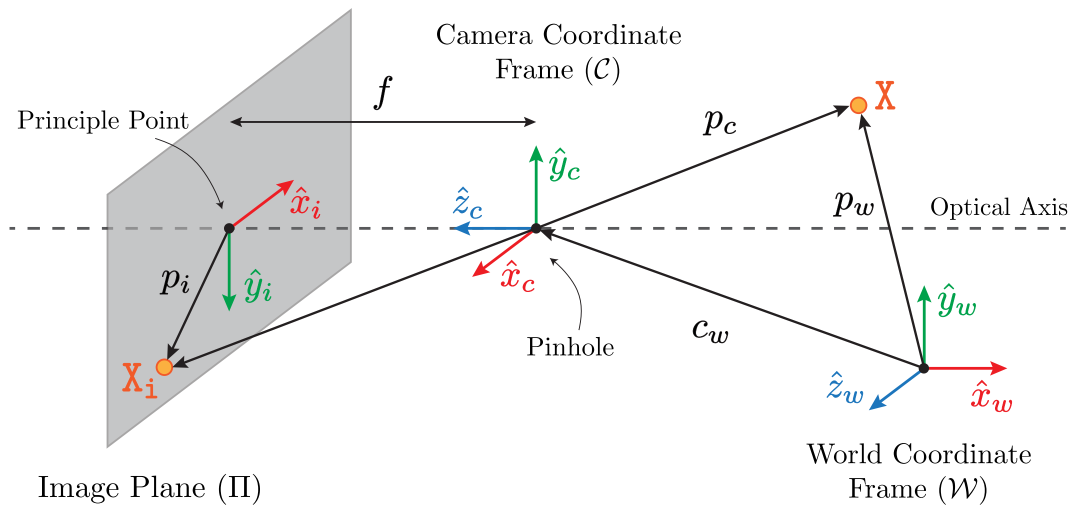


Figure 4.1: Pinhole camera model.

The construction of the pinhole camera model involves defining the relationships between the 3D world, the pinhole camera, and the resulting 2D image. To reflect these 4 different frames of reference, all of which :

- **World Coordinate Frame ($\mathcal{W}$).** 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 ($\mathcal{C}$).** Represents the 3D space of the scene being photographed, but with respect to the pinhole (aperture) of the camera.
- **Image Coordinate Frame (Π).** 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. The Discrete version of the image coordinate frame, where c

The optical axis of a camera is an imaginary line which passes through the center of the aperture of the camera.

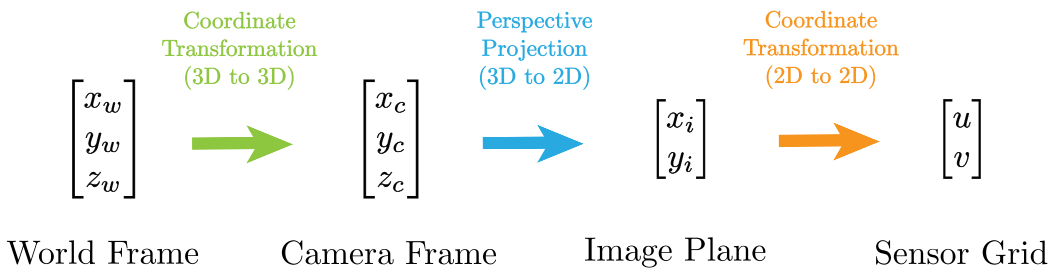

Coordinate Transformation (3D to 3D)
Perspective Projection (3D to 2D)
Coordinate Transformation (2D to 2D)
$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$
$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$
$\begin{bmatrix} x_i \\ y_i \end{bmatrix}$
$\begin{bmatrix} u \\ v \end{bmatrix}$
World Frame
Camera Frame
Image Plane
Sensor Grid

Figure 4.2: Coordinate transformations.

4.1 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 the point `X` to its projection on the image plane.

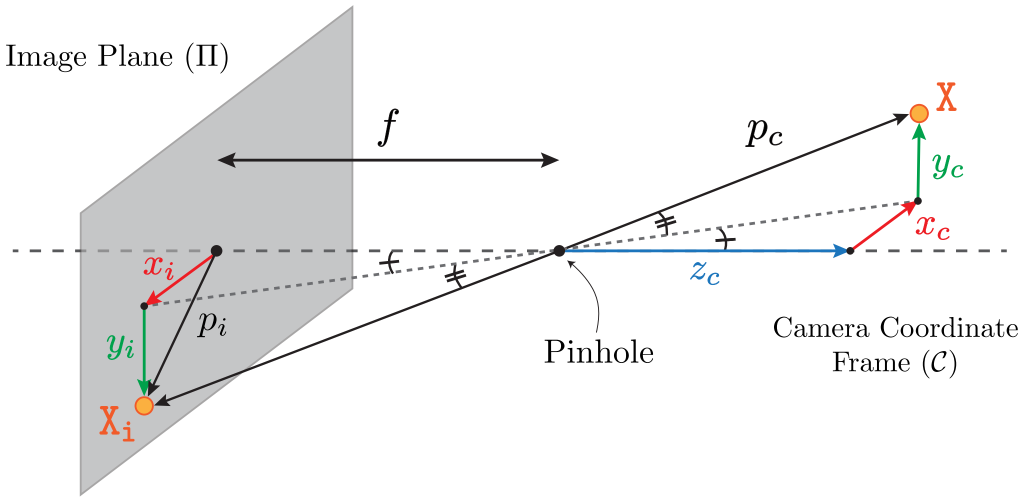


Figure 4.3: Perspective projection of the point `X` onto the image plane Π.

When a straight line is drawn from `X` to its projection `Xi` 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.

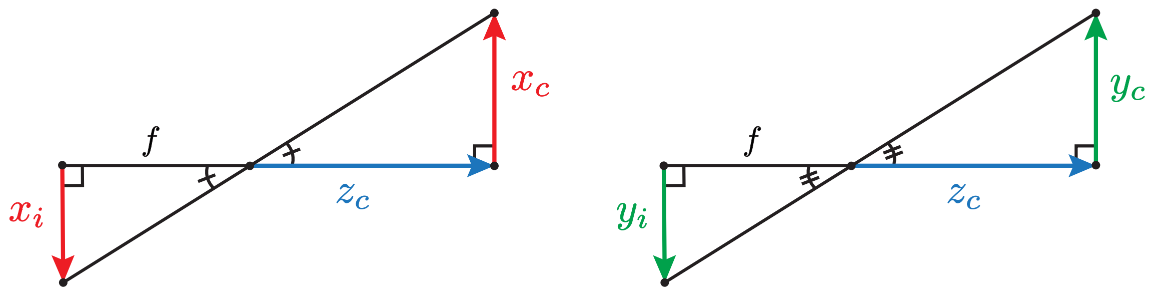


Figure 4.4: Similar triangles formed by perspective projection, which relate x_i to x_c and y_i to y_c.

$$\frac{x_i}{f} = \frac{x_c}{z_c} \implies x_i = f\frac{x_c}{z_c} \tag{4.1a}$$

$$\frac{y_i}{f} = \frac{y_c}{z_c} \implies y_i = f\frac{y_c}{z_c} \tag{4.1b}$$

Once the coordinates of the point projection, (x_i, y_i), is known, we then need to convert it to actual pixel position of the point on the image, (u, v). 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, c_y), which represents the optical center of the image (i.e. the point at which the optical axis intersects the image plane). Additionally, the relationship between (x_i, y_i) and (u, v) is proportional, but they scale at different rates, as (x_i, y_i) can be measured using any unit measurement, and can have negative and decimal values. On the other hand, $(u$,$v)$ are measured in discrete pixel value, which can different sizes depending on the camera used. As such, we define scaling factors, m_x and m_y, be scaling factors which represent the pixel density of the image sensor in the x and y axes of the image sensor plane respectively.

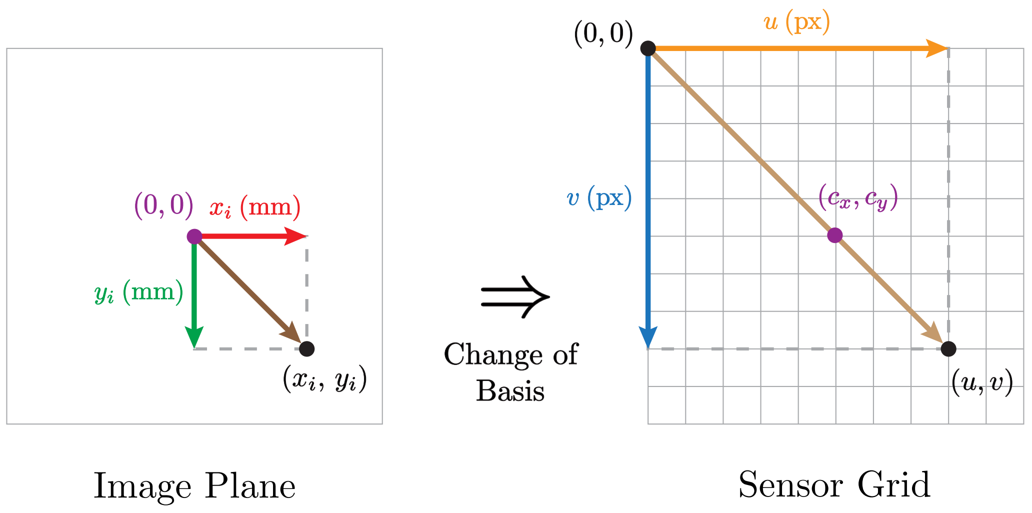


Figure 4.5: Conversion from image plane coordinates to sensor grid coordinates

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 4.1a and 4.1b, 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.

$$u = f_x \frac{x_c}{z_c} + c_x \tag{4.2}$$

$$v = f_y \frac{y_c}{z_c} + c_y \tag{4.3}$$

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 homogenous coordinates, by letting $\widetilde{w} = z_c$.

$$\begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c c_x \\ f_y y_c + z_c c_y \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{4.5}$$

This can be represented simply with:

$$\widetilde{p_i} = mK\, p_c\,, \qquad m \in \mathbb{R}^+ \tag{4.6}$$

In this case, K is what 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.

4.2 Extrinsic Parameters

Extrinsic parameters describe the orientation of the camera. As such, they describe 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:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.7}$$

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.

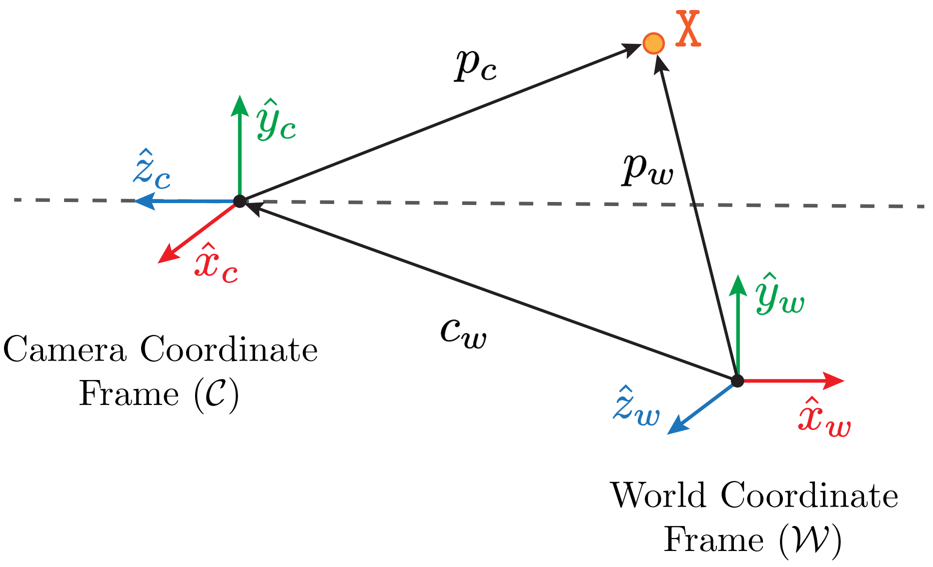


Figure 4.6: Coordinate transformation of X from the world coordinate frame to the camera coordinate frame.

From Figure 4.6, we can see that the naive approach to finding the position vector of the point in $\mathcal{C}$ p_c is equal to the position vector of the point in $\mathcal{W}$ p_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:

$$\begin{aligned} p_c &= R\,(p_w - c_w) \\ &= R\,p_w - R\,c_w \end{aligned} \tag{4.8}$$

Since the position of the camera c_w is constant, we can replace the constant vector $-R\,c_w$ with t, and in this case t represents the translation of the camera from the origin.

$$p_c = R\,p_w + t \tag{4.9}$$

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

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R\,|\,t\,]} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{4.10}$$

Thus, we arrive at the final equation:

$$p_c = [R\,|\,t\,]\,\widetilde{p}_w \tag{4.11}$$

5 Projection Matrix

When we combine the equation for the intrinsic transformation, $k\,\widetilde{p}_i = K\,p_c$ (eq. 4.6), with the equation for the extrinsic transformation, $p_c = [R\,|\,t\,]\,\widetilde{p}_w$ (eq. 4.11), we obtain:

$$\widetilde{p}_i = mK\,[R\,|\,t\,]\,\widetilde{p}_w\,, \qquad m \in \mathbb{R}^+ \tag{5.1}$$

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.

$$\underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{P} \equiv \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R\,|\,t\,]} \tag{5.2}$$

Replacing P for $K\,[R\,|\,t\,]$ in equation 5.1, we obtain:

$$\widetilde{p}_i = mP\,\widetilde{p}_w\,, \qquad m \in \mathbb{R}^+ \tag{5.3}$$

Equivalently:

$$\begin{bmatrix} \widetilde{u}_n \\ \widetilde{v}_n \\ \widetilde{w}_n \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(n)} \\ y_w^{(n)} \\ z_w^{(n)} \\ 1 \end{bmatrix} \tag{5.4}$$

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.

5.1 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 projections. Rewriting the matrix equation 5.3 as a system of equations, we obtain:

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

We convert the set of equations back to their inhomogeneous form by dividing $\widetilde{u}_n$ and $\widetilde{v}_n$ by $\widetilde{w}_n$.

$$\begin{aligned}u_n &= \frac{\widetilde{u}_n}{\widetilde{w}_n} = \frac{p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14}}{p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}}\\ v_n &= \frac{\widetilde{u}_n}{\widetilde{w}_n} = \frac{p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24}}{p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}}\end{aligned}$$

For both equations, multiply both sides by the denominator.

$$\begin{aligned}u_n(p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}) &= p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14}\\ v_n(p_{31}x_w^{(n)} + p_{32}y_w^{(n)} + p_{33}z_w^{(n)} + p_{34}) &= p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24}\end{aligned}$$

Bring all the terms onto one side:

$$0 = p_{11}x_w^{(n)} + p_{12}y_w^{(n)} + p_{13}z_w^{(n)} + p_{14} - p_{31}u_nx_w^{(n)} - p_{32}u_ny_w^{(n)} - p_{33}u_nz_w^{(n)} - p_{34}u_n \tag{5.5a}$$

$$0 = p_{21}x_w^{(n)} + p_{22}y_w^{(n)} + p_{23}z_w^{(n)} + p_{24} - p_{31}v_nx_w^{(n)} - p_{32}v_ny_w^{(n)} - p_{33}v_nz_w^{(n)} - p_{34}v_n \tag{5.5b}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\ & & \vdots & & & & & & & \vdots & & \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}}_{p} \tag{5.6}$$

where G is a $2n \times 12$ matrix, $n \in \mathbb{Z}^+$, and p is the matrix vectorization of P. This system has 12 degrees of freedom, and since each point yields two equations, a minimum of 6 sets of points and their corresponding image projections are necessary in order to solve the system. 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 such systems generally have no solution[9]. However, we can optimize to obtain a best-fit solution using a method such as the *Constrained Least Squares Solution*.

5.2 Constrained Least Squares Solution

We have now established a way to solve for the

[9]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.

Now, we need to solve for $Gp = 0$

$$\underset{p}{\text{minimize}} \;\; \|Gp\|^2 \;\; \text{subject to} \;\; \|p\|^2 = 1 \tag{5.7}$$

For a given arbitrary vector $v \in \mathbb{R}^n$, the magnitude is equal to $\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$. As such, we can rewrite the square of the magnitude of v, $\|v\|^2$, as:

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

Thus, in equation 5.7, we can replace $\|Gp\|^2$ with $p^\mathsf{T} A^\mathsf{T} Gp$ and $\|p\|^2$ for $p^\mathsf{T} p$ to obtain

$$\underset{p}{\text{minimize}} \;\; \left(p^\mathsf{T} G^\mathsf{T} Gp\right) \;\; \text{subject to} \;\; p^\mathsf{T} p = 1 \tag{5.8}$$

The Lagrangian[10] of equation 5.8 is

$$\mathcal{L}(p, \lambda) = p^\mathsf{T} G^\mathsf{T} Gp - \lambda \left(p^\mathsf{T} p - 1\right) \tag{5.9}$$

where $\lambda \in \mathbb{R}$ is the Lagrange multiplier. Since p is minimized when $\mathcal{L}$ is minimized, we need to look for the absolute minimum of $\mathcal{L}$, which are located at its critical points. To find these points, we want to look for values of p and λ where all partial derivatives of the Lagrangian are zero, i.e.

$$\frac{\partial \mathcal{L}}{\partial p} = 0 \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial \lambda} = 0$$

where ∂ is used to denote a partial derivative. We will focus on the partial derivative of $\mathcal{L}$

[10] 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]`.

with respect to p. Using product rule for partial derivatives, we obtain:

$$\frac{\partial \mathcal{L}}{\partial p} = \frac{\partial}{\partial p}\left[p^{\mathsf{T}}G^{\mathsf{T}}Gp - \lambda\left(p^{\mathsf{T}}p - 1\right)\right] \overset{\text{set}}{=} 0$$

$$\Rightarrow 2G^{\mathsf{T}}Gp - 2\lambda p = 0$$

$$\Rightarrow G^{\mathsf{T}}Gp = \lambda p \tag{5.10}$$

which is an eigenvalue problem for $G^{\mathsf{T}}G$. Potential solutions for p are eigenvectors that satisfy equation 5.10,[11] with $\lambda \in \mathbb{R}$ as the eigenvalue. Since 5.8 is a minimization problem, the minimized eigenvector p is the one which has the smallest eigenvalue λ.[12]

which states that for a given matrix $M \in \mathbb{R}^{n\times n}$, determine the eigenvector $x \in \mathbb{R}^n, x \neq 0$ and the eigenvalue $\lambda \in \mathbb{C}$ such that:

6 Extracting Parameters

Once we have solved for the projection for the projection matrix P, we can then extract the intrinsic and extrinsic parameters. We know that

An important property worth noting is that K is an *upper triangular matrix*. It is a special kind of square matrix with all of its non-zero entries above the main diagonal. This is an important property which we will exploit when extracting the K from the projection matrix in section 5.

[11] Shree Nayar, ed., *Linear Camera Model*, April 18, 2021, accessed August 23, 2023, `https://www.youtube.com/watch?v=qByYk6JggQU`, accessed August 23, 2023, `https://www.youtube.com/watch?v=qByYk6JggQU`.

[12] Ghojogh, Karray, and Crowley, "Eigenvalue and Generalized Eigenvalue Problems."

$$\begin{aligned} P &= K\,[\,R \mid t\,] \\ &= K\,[\,R \mid -Rc_w\,] \\ &= [\,KR \mid -KRc_w\,] \\ &= [\,Q \mid -Qc_w\,], \qquad \text{with } Q = KR \end{aligned} \tag{6.1}$$

6.1 RQ Decomposition

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

Since K is in the form of an *upper right triangular matrix* and R is an *orthonormal matrix*, we can find unique solutions for K and R using a method called *RQ decomposition.*

RQ decomposition is a technique which allows us to uniquely decompose a matrix A into a product $A = RQ$, where R is an upper

Since

6.2 Extracting the Translation Vector

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

$$\Rightarrow c_w = -Q^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} \tag{6.2}$$

6.3 Extracting Orientation as Angles

When constructing the extrinsic matrix in section 4.2, we defined the rotation of the camera as a 3×3 matrix, where the

We can represent the rotation in terms of *Tait-Bryan Angles*, where the rotation is represented as 3 elemental rotations about each of the principle axes.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{6.3a}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & -\cos(\beta) \end{bmatrix} \tag{6.3b}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3c}$$

$$R \equiv R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{6.4}$$

$$\begin{aligned} R &= \begin{bmatrix} 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & -\cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\beta)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\cos(\gamma) \\ \cos(\beta)\sin(\gamma) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) \\ -\sin(\beta) & \sin(\alpha)\cos(\beta) & \cos(\alpha)\cos(\beta) \end{bmatrix} \end{aligned} \tag{6.5}$$

We have that

$$r_{31} = -\sin(\beta)$$

$$\Rightarrow \beta = \sin^{-1}(-r_{31}) \tag{6.6}$$

$$r_{32} = \sin(\alpha)\cos(\beta)$$

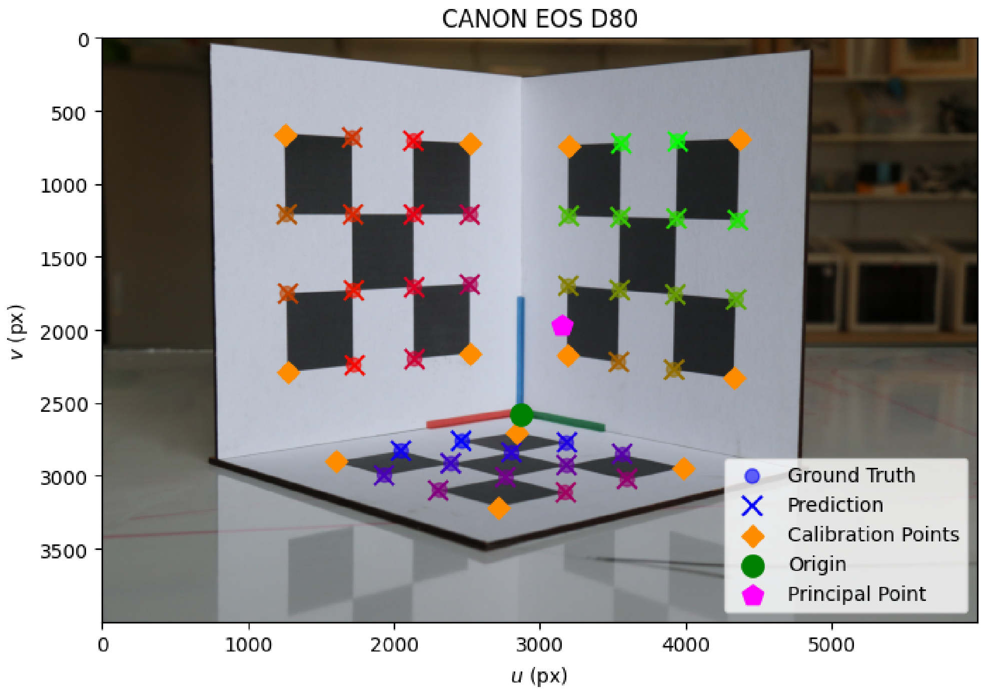
$$\begin{aligned}\Rightarrow \alpha &= \sin^{-1}\left(\frac{r_{32}}{\cos(\beta)}\right) = \sin^{-1}\left(\frac{r_{32}}{\cos(\sin^{-1}(-r_{31}))}\right) \\ &= \sin^{-1}\left(\frac{r_{32}}{\sqrt{1-r_{31}^2}}\right)\end{aligned} \tag{6.7}$$

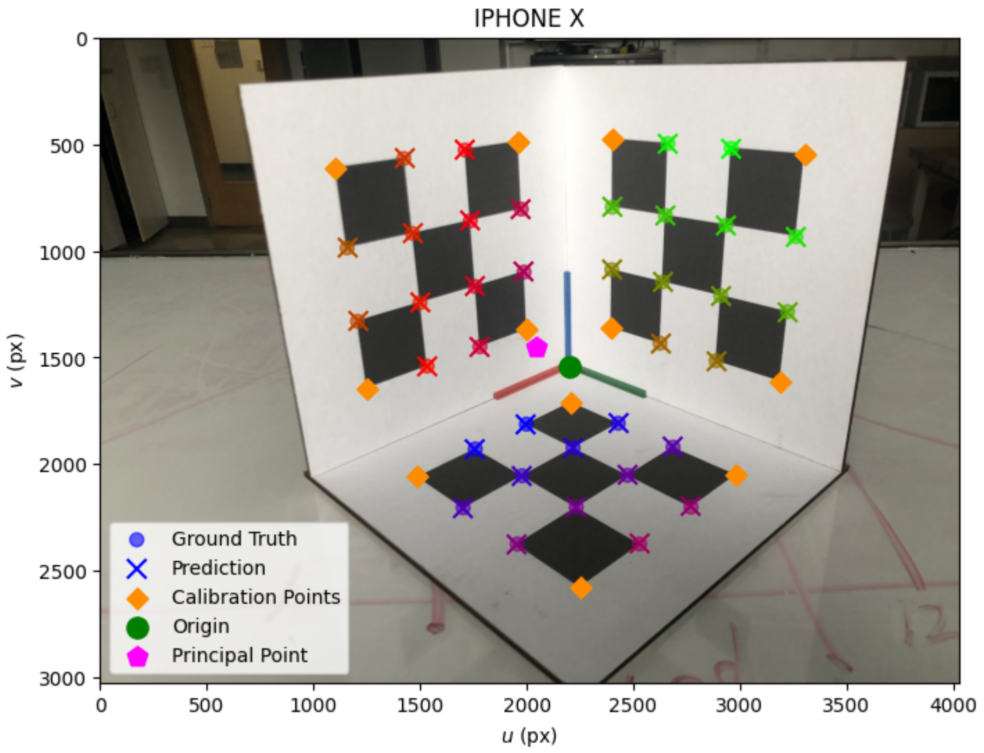
$$r_{21} = \cos(\beta)\sin(\gamma)$$

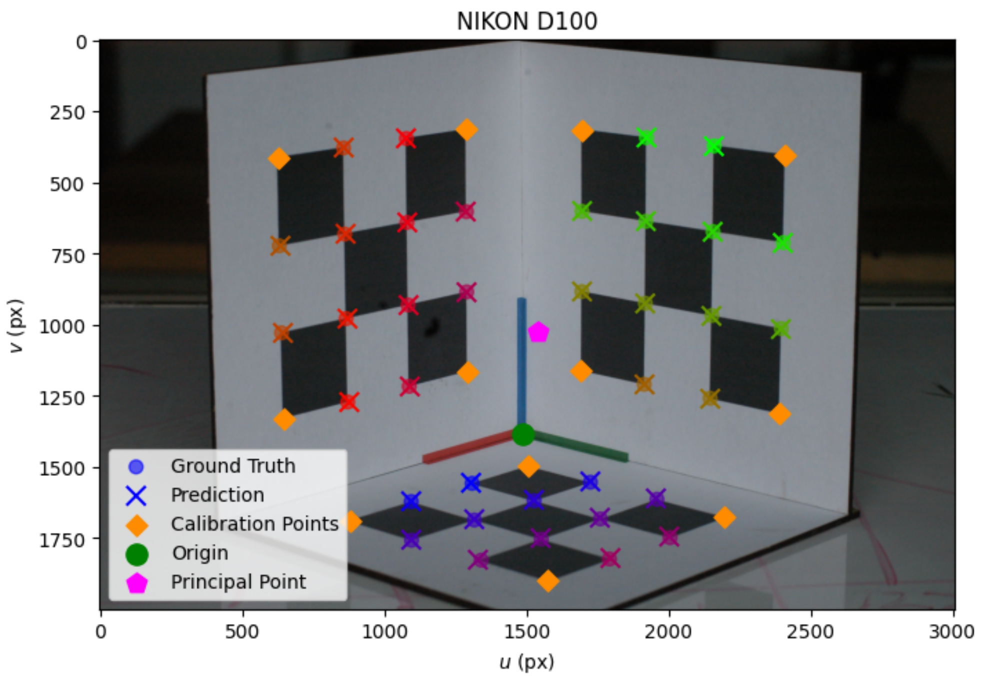
$$\begin{aligned}\Rightarrow \gamma &= \sin^{-1}\left(\frac{r_{21}}{\cos(\beta)}\right) = \sin^{-1}\left(\frac{r_{21}}{\cos(\sin^{-1}(-r_{31}))}\right) \\ &= \sin^{-1}\left(\frac{r_{21}}{\sqrt{1-r_{31}^2}}\right)\end{aligned} \tag{6.8}$$

7 Experimental Validation

In an attempt to show that the model works, I created the program


CANON EOS D80
v (px)
u (px)
Ground Truth
Prediction
Calibration Points
Origin
Principal Point


IPHONE X
v (px)
u (px)
Ground Truth
Prediction
Calibration Points
Origin
Principal Point



$$P = \begin{bmatrix} -2.5844 \times 10^{-3} & 1.7334 \times 10^{-3} & -4.6719 \times 10^{-4} & 6.0581 \times 10^{-1} \\ 4.8240 \times 10^{-4} & 4.4097 \times 10^{-4} & -3.1337 \times 10^{-3} & 7.9559 \times 10^{-1} \\ -3.3990 \times 10^{-7} & -3.1311 \times 10^{-7} & -2.8179 \times 10^{-7} & 4.1340 \times 10^{-4} \end{bmatrix}$$

		Canon EOS D80	**IPhone X**	**Nikon D100**
Focal Lengths	f_x	8404.1 px	3281.5 px	8144.4 px
	f_y	8387.9 px	3279.9 px	8142.6 px
Principal Point	c_x	3151.6 px	2043.0 px	1541.8 px
	c_y	1972.8 px	1453.1 px	1027.9 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 7.1: Intrinsic and Extrinsic Parameters calculated by `calicam`.

7.1 Validating Estimated Focal Length

Given that specification of cameras are readily available online, we can actually evaluate the accuracy of our calculated focal lengths. 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 [13]	**Manufacturer Reported Focal Length**	**% Error**
Canon EOS D80	$(8496\,\text{px})(3.73\,\mu\text{m/px}) \approx \boxed{31.7\,\text{mm}}$	32 mm	0.94 %
IPhone X	$(3280\,\text{px})(1.22\,\mu\text{m/px}) \approx \boxed{4.00\,\text{mm}}$	4 mm	–
Nikon D100	$(8143\,\text{px})(7.82\,\mu\text{m/px}) \approx \boxed{63.7\,\text{mm}}$	55 mm	15.6 %

Table 7.2: Comparison of Calculated vs. Reported Focal Length.

Considering that the focal lengths reported by manufacturers are often only accurate to around ± 1 mm,[14] 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 Nikon D100 altered the effective focal length.

8 Conclusion

From my rudimentary experimental validation, we can see that my devised method of camera calibration is very accurate, and despite its various limitations, such as fixed focal length and fixed locality, it serves as a proof of concept which excellently demonstrates the fundamental techniques behind camera calibration.

Acknowledgements

I am very grateful to my supervisor Mr. Hoteit 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 Mr. Matthewson for his guidance with the manufacturing of my calibration object, and to Leon, for his help in operating the laser cutter. Last but not least, I want to thank Aditya for aiding me in the process of creating some of the diagrams used in my paper.

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

[14] 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.

Bloomenthal, Jules, and Jon Rokne. “Homogeneous Coordinates.” *The Visual Computer* 11, no. 1 (January 1994): 15–26. Accessed October 20, 2023. http://link.springer.com/10.1007/BF01900696. 10.1007/BF01900696.

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/.

Nayar, Shree, ed. *Linear Camera Model*. April 18, 2021. Accessed August 23, 2023. https://www.youtube.com/watch?v=qByYk6JggQU.

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`.

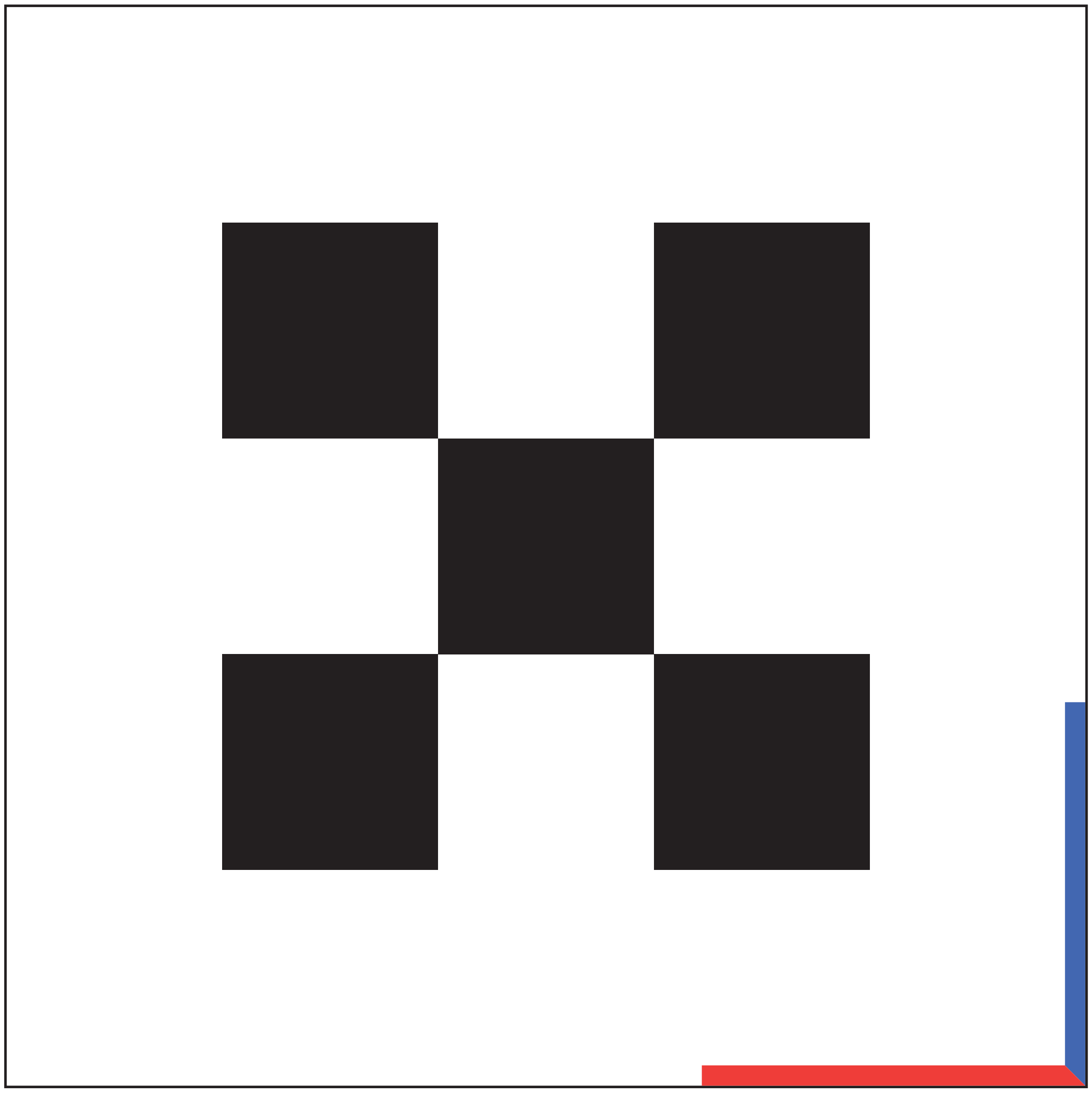
Williams, Gareth. “Overdetermined Systems of Linear Equations.” *The American Mathematical Monthly* 97, no. 6 (June 1990): 511–513. Accessed November 1, 2023. `https://www.jstor.org/stable/2323837`. JSTOR: `2323837`.

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`.

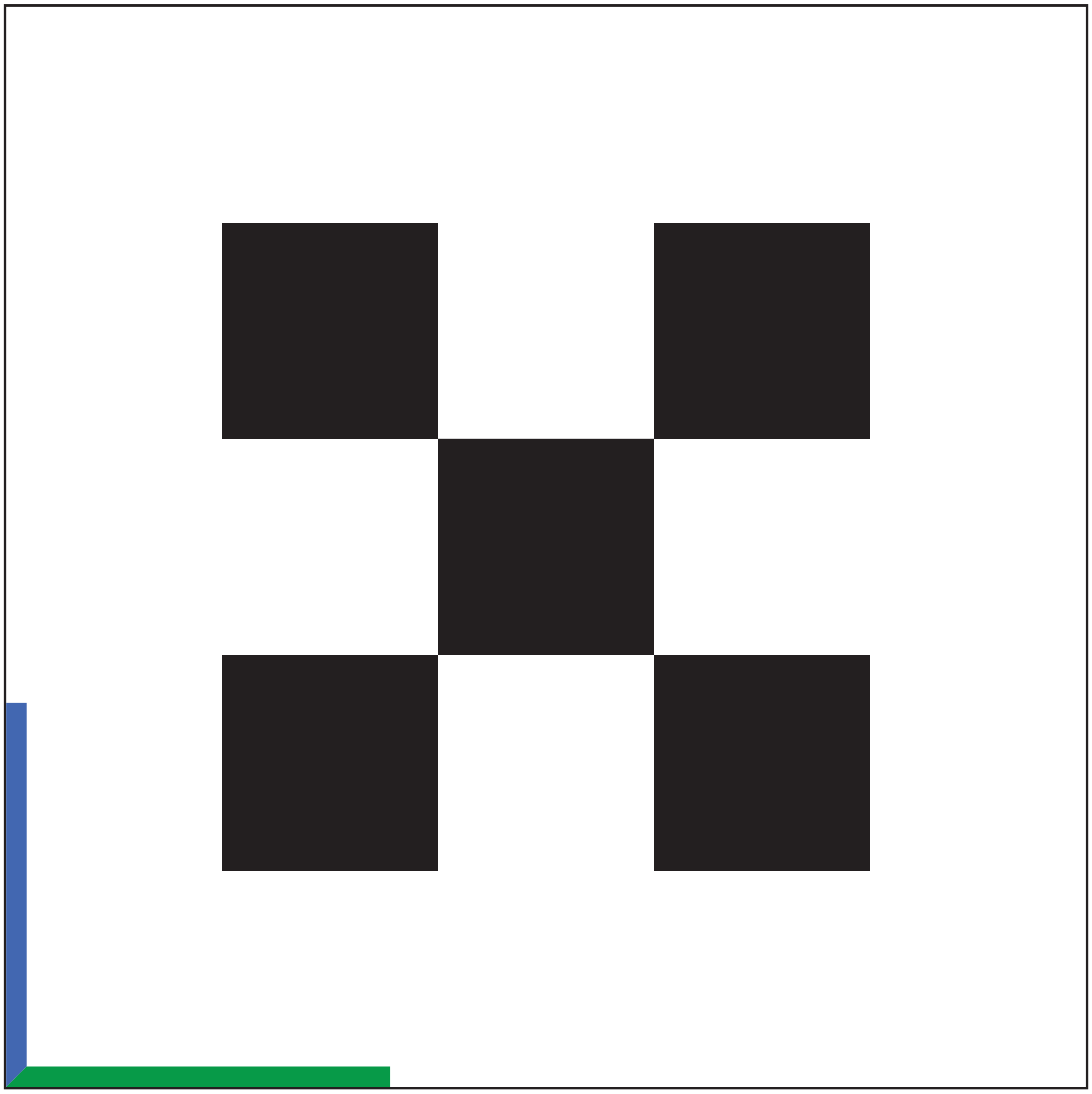
———. “Camera Calibration,” May 2007. Accessed October 10, 2023. `https://people.cs.rutgers.edu/elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf`.

Appendix A Calibration Object Panel Patterns

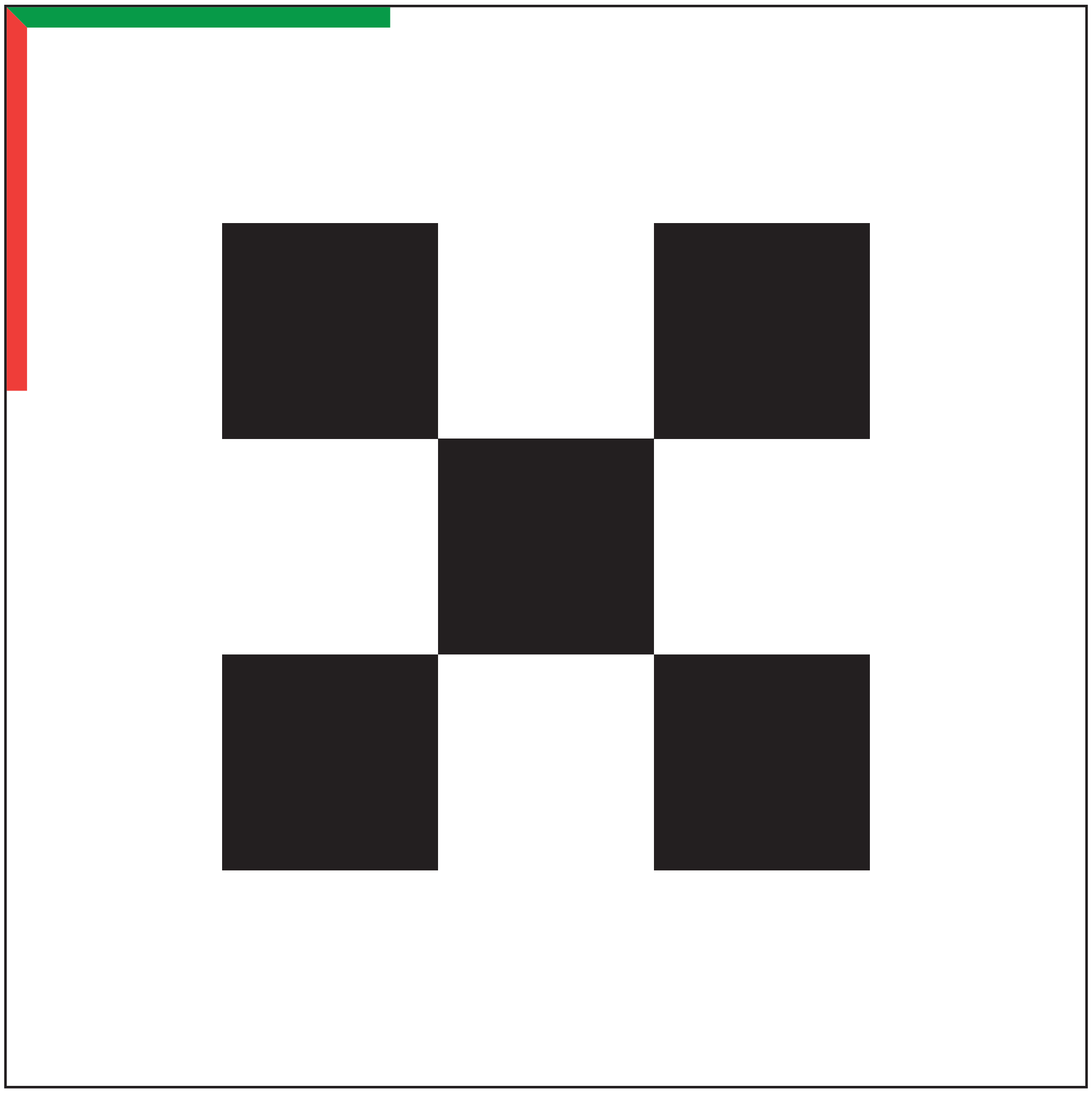
XZ Face



YZ Face



XY Face



Appendix B 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

        cali_world_coords, cali_image_coords = calicam.parse_data_from_csv(csv_path)
        proj_matrix, cali_matrix, rot_matrix, (tx, ty, tz) = calicam.calibrate_camera(
            cali_world_coords, cali_image_coords)

        # origin
        (ox, oy) = calicam.project_point(proj_matrix, (0.0, 0.0, 0.0))

        # principal point, focal lengths
        (cx, cy), (fx, fy) = calicam.extract_intrinsics(cali_matrix)

        # tait-bryan angles
        a, b, g = calicam.extract_orientation_zyx(rot_matrix)

        output.append("\n" + "\n\n".join((
            f"Projection Matrix: \n{proj_matrix}",
            f"Calibration Matrix: \n{cali_matrix}",
            f"Rotation Matrix: \n{rot_matrix}",
            f"Focal Lengths: \n\tf_x = {fx:.2f} px \n\tf_y = {fy:.2f} px",
            f"Principal Point: \n\tc_x = {cx:.2f} px \n\tc_y = {cy:.2f} px",
            f"Translation: \n\tt_x = {tx:.2f} \n\tt_y = {ty:.2f} \n\tt_z = {tz:.2f}",
            f"Orientation: \n\t\u03B1 = {a:.2f}° \n\t\u03B2 = {b:.2f}° \n\t\u03B3 = {g:.2f}°",
        )))

        # MODEL VALIDATION
        data_path: str | None = args.data

        if data_path is not None:
            assert os.path.isfile(data_path), f"{data_path} does not exist."
            assert data_path.endswith(".csv"), f'{data_path} does not end with the extension ".csv".'

            data_world_coords, data_image_coords = calicam.parse_data_from_csv(data_path)

            predicted_coords = [
                calicam.project_point(proj_matrix, world_coord) for world_coord in data_world_coords
            ]
            reproj_errs = [
                calicam.euclidean(actual_coord, reproj_coord)
                for actual_coord, reproj_coord in zip(data_image_coords, predicted_coords)
            ]

            max_err = max(reproj_errs)
            avg_err = sum(reproj_errs) / len(reproj_errs)

            output.append(
                f"\nReprojection Errors: \n\t\u03BC_max = {max_err:.3f} px \n\t\u03BC_avg = {avg_err:.3f}
                ↪  px")

        # OUTPUT
        if not args.noprint:
            print(*output, sep="\n")

        # GRAPH
        image_path: str | None = args.graph

        if image_path is not None:
            ax: plt.Axes
            _, ax = plt.subplots(figsize=(8, 10))

            plt.gca().invert_yaxis()

            # image was provided
            if image_path != "":
                assert os.path.isfile(image_path), f"{image_path} does not exist."
                assert args.data, "Path to data csv file must be provide using -d flag to produce graph."

                img = plt.imread(image_path,)
                ax.imshow(img, cmap='gray')
                ax.autoscale(False)

            # graph data points and model points only if -d flag is specfied
            if data_path is not None:
                cmap = plt.cm.brg
                discrete_cmap = list(cmap(np.linspace(0, 1, len(data_image_coords))))

                # data points
                ax.scatter(
                    *zip(*data_image_coords),
                    label="Ground Truth",
                    s=50,
                    color=discrete_cmap,
                    marker="o",
                    alpha=0.6,
                )

                # predicted points
                ax.scatter(
                    *zip(*predicted_coords),
                    label="Prediction",
                    s=100,
                    color=discrete_cmap,
                    marker="x",
                )

            # calibration points
            ax.scatter(
                *zip(*cali_image_coords),
                label="Calibration Points",
                s=60,
                marker="D",
                color="darkorange",
            )

            # origin point
            ax.scatter(ox, oy, label="Origin", s=120, marker="o", color="green")

            # principle point
            ax.scatter(cx, cy, label="Principal Point", s=120, marker="p", color="magenta")

            graph_title: str = args.title or image_path

            plt.gca().update({"title": graph_title, "xlabel": "$u$ (px)", "ylabel": "$v$ (px)"})
            plt.legend()

            out_path: str | None = args.out

            if out_path:
                plt.savefig(out_path, bbox_inches='tight')

            # show graph if -s flag was specified or if a save location was not specified
            if args.show or not out_path:
                plt.show()

    except AssertionError as e:
        parser.error(str(e))  # pass error to argparse

    except KeyboardInterrupt:
        print(f"\nKeyboardInterrupt")
        sys.exit(1)

    sys.exit(0)


if __name__ == "__main__":
    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

```python
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

```python
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)))
```