

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: 2762 words

Contents

1	Introduction	1
1.1	Problem Statement	1
2	Approach	2
2.1	Camera Model	2
2.1.1	Pinhole Camera Model	3
2.2	Calibration Object	4
3	Prerequisites	5
3.1	Notation	5
3.2	Homogenous Coordinates	6
4	Constructing the Pinhole Camera Model	7
4.1	Intrinsic Parameters	8
4.2	Extrinsic Parameters	11
5	Projection Matrix	13
5.1	Solving for the Projection Matrix	14
5.2	Constrained Least Squares Solution	16
6	Extracting Parameters	18
6.1	RQ Decomposition	18
6.2	Extracting the Translation Vector	19
6.3	Extracting Orientation as Angles	19
7	Experimental Validation	20
7.1	Validating Estimated Focal Length	23
8	Conclusion	24
	Acknowledgements	24
	Bibliography	25

A Calibration Object Panel Patterns	27
B Source Code	30

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 drawings¹. 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

¹Joerg 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.

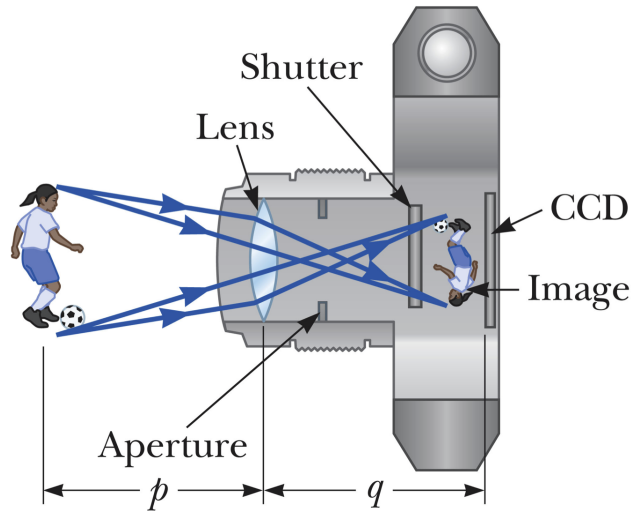


Figure 2.1: Lens camera. Adapted from John Colton, “Physics 123 Lecture 30 Warm-up Questions,” BYU Physics and Astronomy, November 5, 2012, accessed October 17, 2023, <https://physics.byu.edu/faculty/colton/docs/phy123-fall12/jitt30a.html>.

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 sensor 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 assumptions that the aperture is infinitely small. This means that any incoming light ray can only travel straight through the pinhole, and that a point in space can only map to one single point on image plane.

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.

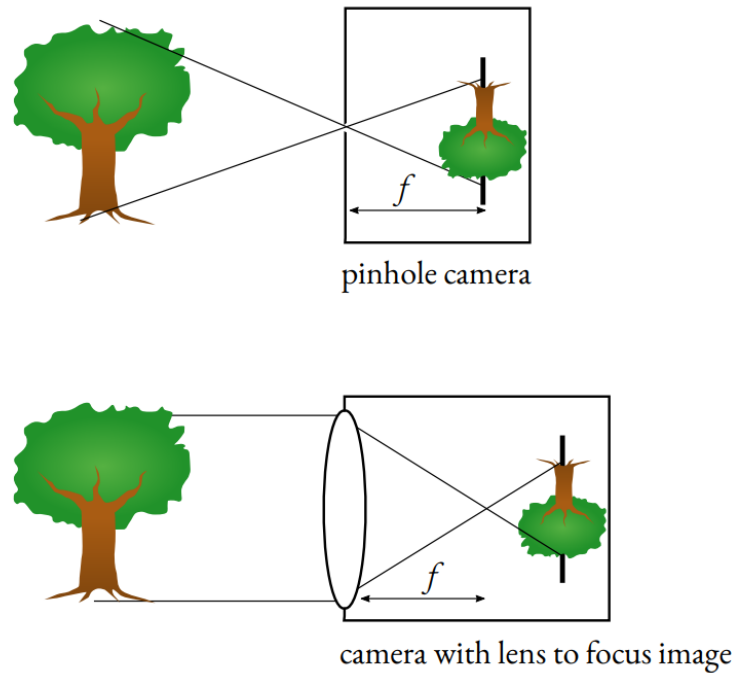


Figure 2.2: Difference between a pinhole camera and a lens camera. Adapted from Hoàng-Ân Lê, “Camera Model: Intrinsic Parameters,” July 30, 2018, accessed October 14, 2023, <https://lhoangan.github.io/camera-params/>.

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 roughly separated into 3 categories, based on the dimension of the calibration object used²:

- **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 points³. Using 3D objects is typically preferred, as it yields the highest accuracy⁴, and the mathematics required is also the least.

²Zhengyou Zhang, “Camera Calibration,” May 2007, accessed October 10, 2023, <https://people.cs.rutgers.edu/elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf>.

³Ibid.

⁴Ibid.

- **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 orientations⁵. 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 library, `OpenCV`, is geared towards this type of calibration.

There also exists other calibration methods, notably self-calibration, which do not require calibration objects and simply rely on image correspondences. Although self-calibration does not require any calibration object, it also means that many parameters need to be estimated and optimized, and as such results in a much more complex mathematical problem⁶.

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 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:

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

⁵Zhengyou 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.

⁶Zhang, “Camera Calibration.”

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 capture the relationships inherent in projective projections and affine transformations, both of which are core concepts in this paper.

Homogenous coordinates form the basis of projective geometry, because it unifies the treatment of common graphical transformations such as rotation and translations⁷.

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

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

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} u\tilde{w} \\ v\tilde{w} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \quad (3.1)$$

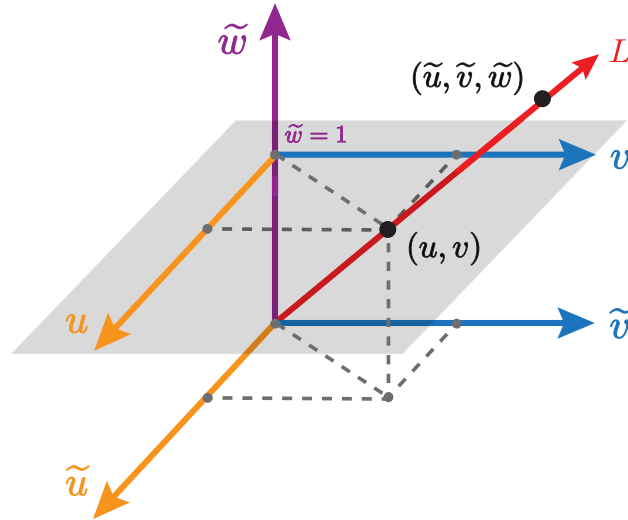


Figure 3.1: Homogenous coordinate system.

⁷Jules 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.

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

4 Constructing the Pinhole Camera Model

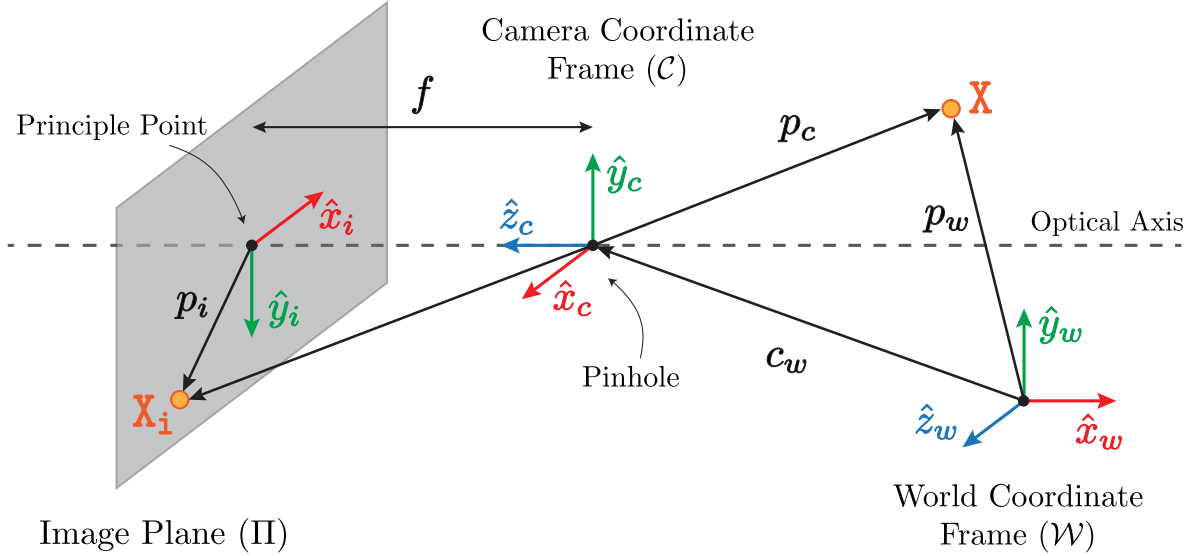


Figure 4.1: Pinhole camera model.

For our camera model, we will introduce 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.

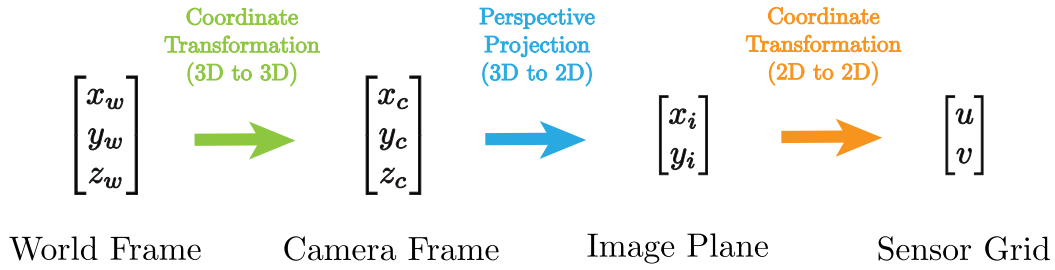


Figure 4.2: Coordinate transformations.

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. As such, our goal is to relate the position of the point X to its projection on the image plane.

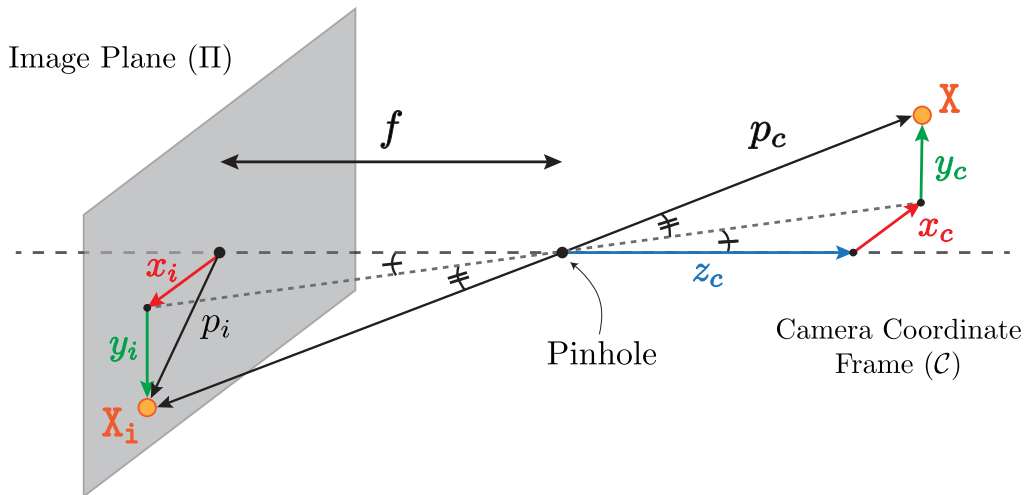


Figure 4.3: Perspective projection of the point X onto the image plane Π .

When a straight line is drawn from \mathbf{X} to its projection \mathbf{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 .

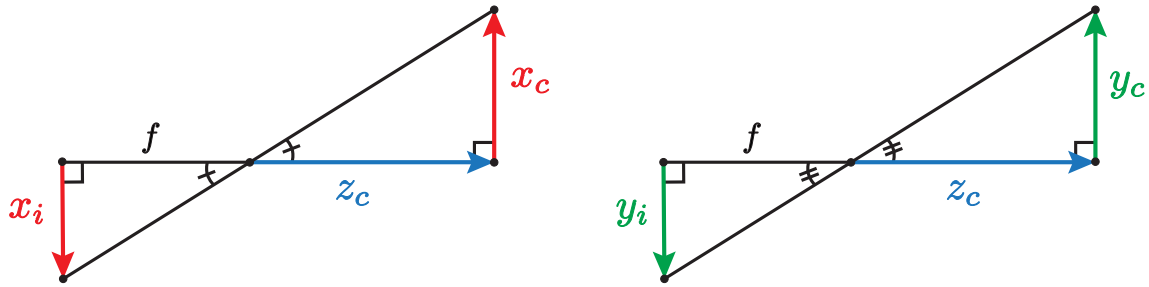


Figure 4.4: Similar triangles formed by perspective projection, which relate x_i to x_c and y_i to y_c .

$$\frac{x_i}{f} = \frac{x_c}{z_c} \implies x_i = f \frac{x_c}{z_c} \quad (4.1a)$$

$$\frac{y_i}{f} = \frac{y_c}{z_c} \implies y_i = f \frac{y_c}{z_c} \quad (4.1b)$$

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.

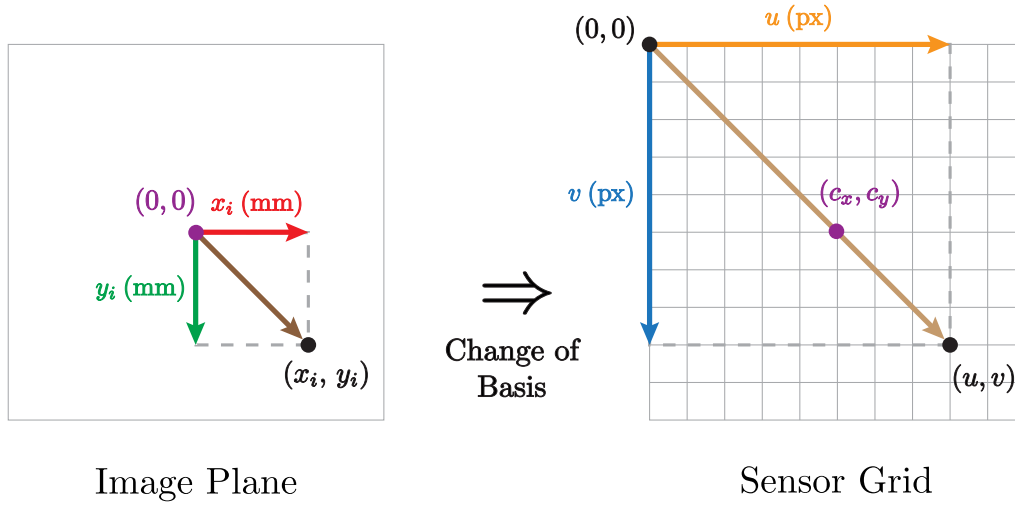


Figure 4.5: Conversion from image plane coordinates to sensor grid coordinates

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 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.

$$u = f_x \frac{x_c}{z_c} + c_x \quad (4.2)$$

$$v = f_y \frac{y_c}{z_c} + c_y \quad (4.3)$$

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 $\tilde{w} = z_c$.

$$\begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c c_x \\ f_y y_c + z_c c_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (4.5)$$

This can be represented simply with:

$$\tilde{p}_i = K p_c, \quad \text{with } K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (4.6)$$

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, which is commonly referred to as the extrinsic parameters of a camera. Possible movements include establish the relationship between position of a point in the camera coordinate frame \mathcal{C} to their position in the world

coordinate frame \mathcal{W} . To do so, we find the extrinsic matrix T , which relates the position of a point \mathbf{X} in world coordinates to its position in camera coordinates. It relates the positional vector p_w of point \mathbf{X} , to its positional vector p_c .

$$p_c = [R \mid t] p_w \quad (4.7)$$

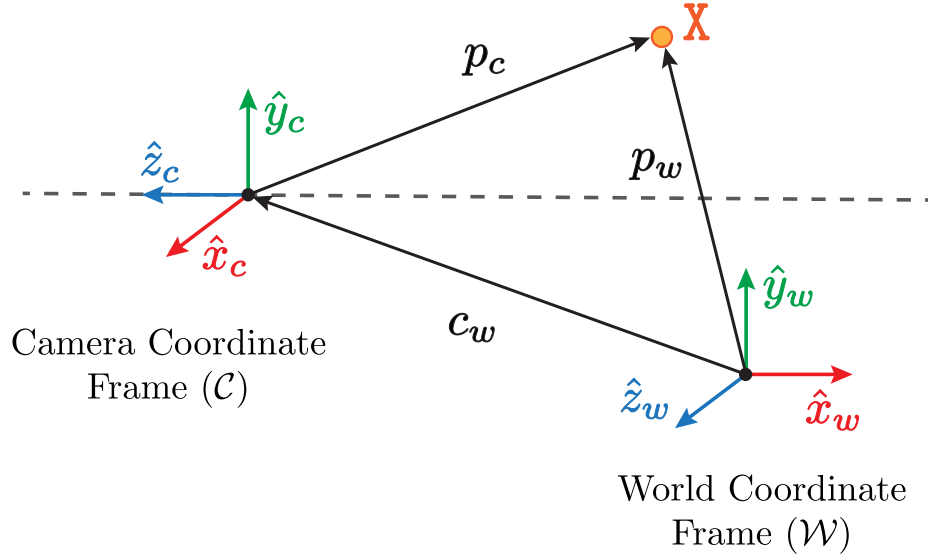


Figure 4.6: Coordinate transformation of \mathbf{X} from \mathcal{W} to \mathcal{C} .

For the extrinsic parameters of the camera, we have the position c_w of the camera in world coordinates and orientation R of the camera. The orientation, R , is a 3x3 rotational matrix:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (4.8)$$

where:

- Row 1: Direction of \hat{x}_c in world coordinate frame.
- Row 2: Direction of \hat{y}_c in world coordinate frame.
- Row 3: Direction of \hat{z}_c in world coordinate frame.

$$p_c = R(p_w - c_w) \quad (4.9a)$$

$$= R p_w - R c_w \quad (4.9b)$$

$$p_c = R p_w + t \quad (4.10)$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \underbrace{\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}_t \quad (4.11)$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R \mid t]} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (4.12)$$

5 Projection Matrix

When we combine the equation for the intrinsic transformation, $p_i = K p_c$ (eq. ??), with the equation for the extrinsic transformation, $p_c = [R \mid t] p_w$ (eq. 4.7), we obtain:

$$k p_i = K T p_w \quad (5.1)$$

This single equation encapsulates the relationship between the world coordinates and its corresponding pixel coordinates. However, since that our goal is to solve for K and T for a particular camera when given the mappings of 3D coordinates to their pixel coordinates, it is much easier to solve for the matrix product $K T$ as opposed to each of the matrices individually. As such, we simplify our camera model by defining a new matrix, P , which is

equal to the product $K T$. Since K is a 3×3 matrix and T is a 3×4 matrix, $K T$ yields a 3×4 matrix.

$$\underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_P \equiv \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_T \quad (5.2)$$

Replacing P for $K T$ in equation 5.1, we obtain:

$$p_i = P p_w \quad (5.3)$$

When expressing the pixel coordinate in homogenous coordinates, equation 5.3 becomes

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

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

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

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

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

We convert the pixel coordinates homogenous coordinates to their inhomogeneous coordinates by dividing \tilde{u}_n and \tilde{v}_n by \tilde{w}_n .

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

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

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

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

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

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

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1 x_w^{(1)} & -u_1 y_w^{(1)} & -u_1 z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1 x_w^{(1)} & -v_1 y_w^{(1)} & -v_1 z_w^{(1)} & -v_1 \\ & & \vdots & & & & & & & \vdots & & \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_n x_w^{(n)} & -u_n y_w^{(n)} & -u_n z_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_n x_w^{(n)} & -v_n y_w^{(n)} & -v_n z_w^{(n)} & -v_n \end{bmatrix}}_G \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} \quad (5.6)$$

$\underbrace{\hspace{10em}}_p$

homogenous linear system overdetermined

5.2 Constrained Least Squares Solution

We have now established a way to solve for the

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

$$\underset{p}{\text{minimize}} \quad \|Gp\|^2 \quad \text{subject to} \quad \|p\|^2 = 1 \quad (5.7)$$

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:

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

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

$$\underset{p}{\text{minimize}} \quad (p^\top G^\top Gp) \quad \text{subject to} \quad p^\top p = 1 \quad (5.8)$$

The Lagrangian⁸ of equation 5.8 is

$$\mathcal{L}(p, \lambda) = p^\top G^\top Gp - \lambda (p^\top p - 1) \quad (5.9)$$

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.

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

where ∂ is used to denote a partial derivative (see Appendix ??). We will focus on the partial derivative of \mathcal{L} with respect to p . Using product rule for partial derivatives, we obtain:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p} &= \frac{\partial}{\partial p} [p^\top G^\top Gp - \lambda (p^\top p - 1)] \stackrel{\text{set}}{=} 0 \\ &\Rightarrow 2G^\top Gp - 2\lambda p = 0 \\ &\Rightarrow G^\top Gp = \lambda p \end{aligned} \quad (5.10)$$

which is an eigenvalue problem for $G^\top G$. Potential solutions for p are eigenvectors that satisfy equation 5.10,⁹ 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 .¹⁰

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:

⁸Benyamin 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].

⁹Shree 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>.

¹⁰Ghojogh, Karray, and Crowley, “Eigenvalue and Generalized Eigenvalue Problems.”

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.

$$\begin{aligned}
 P &= K [R \mid t] \\
 &= K [R \mid -Rc_w] \\
 &= [KR \mid -KRc_w]
 \end{aligned} \tag{6.1}$$

$$P = [Q \mid -Qc_w] \tag{6.2}$$

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

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*.

6.1 RQ Decomposition

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

Since

6.2 Extracting the Translation Vector

$$\begin{aligned}
 -Qc_w &= \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} \\
 \Rightarrow c_w &= -Q^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}
 \end{aligned} \tag{6.3}$$

6.3 Extracting Orientation as Angles

When constructing the extrinsic matrix in section 4.2, we defined the rotation matrix as the

We can represent the rotation in terms of *Tait-Bryan Angles*

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

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{6.5a}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{6.5b}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.5c}$$

$$\begin{aligned}
R &= \begin{bmatrix} 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & -\cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos(\beta) \cos(\gamma) & \sin(\alpha) \sin(\beta) \cos(\gamma) - \cos(\alpha) \sin(\gamma) & \cos(\alpha) \sin(\beta) \cos(\gamma) + \sin(\alpha) \cos(\gamma) \\ \cos(\beta) \sin(\gamma) & \sin(\alpha) \sin(\beta) \sin(\gamma) + \cos(\alpha) \cos(\gamma) & \cos(\alpha) \sin(\beta) \sin(\gamma) - \sin(\alpha) \cos(\gamma) \\ -\sin(\beta) & \sin(\alpha) \cos(\beta) & \cos(\alpha) \cos(\beta) \end{bmatrix} \quad (6.6)
\end{aligned}$$

We have that

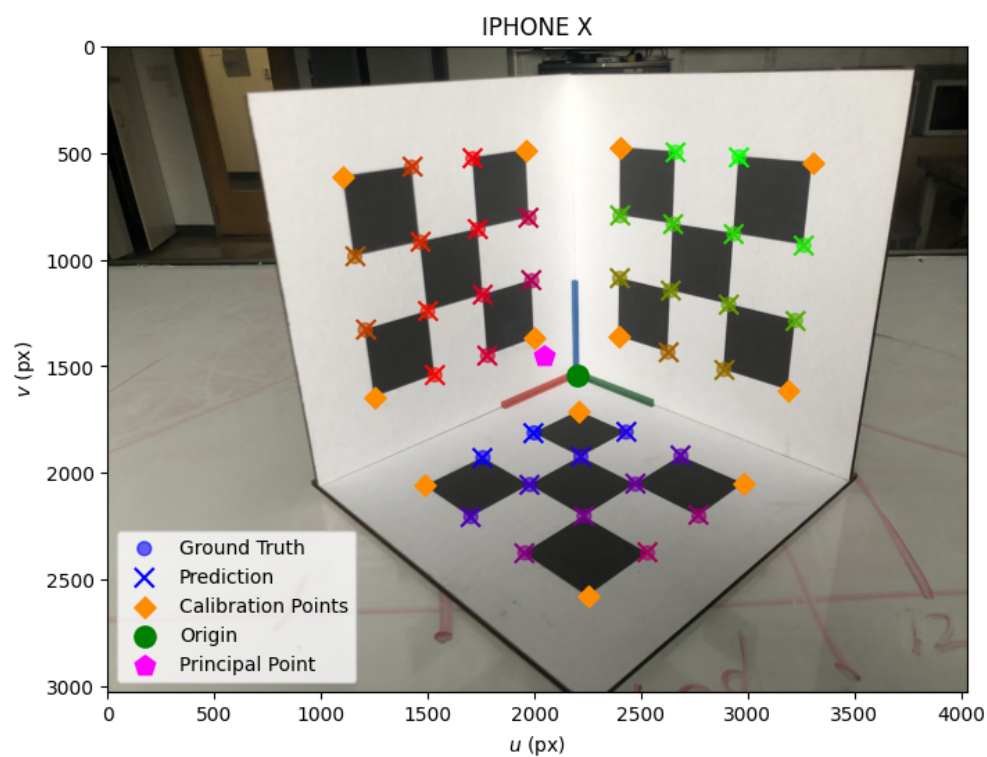
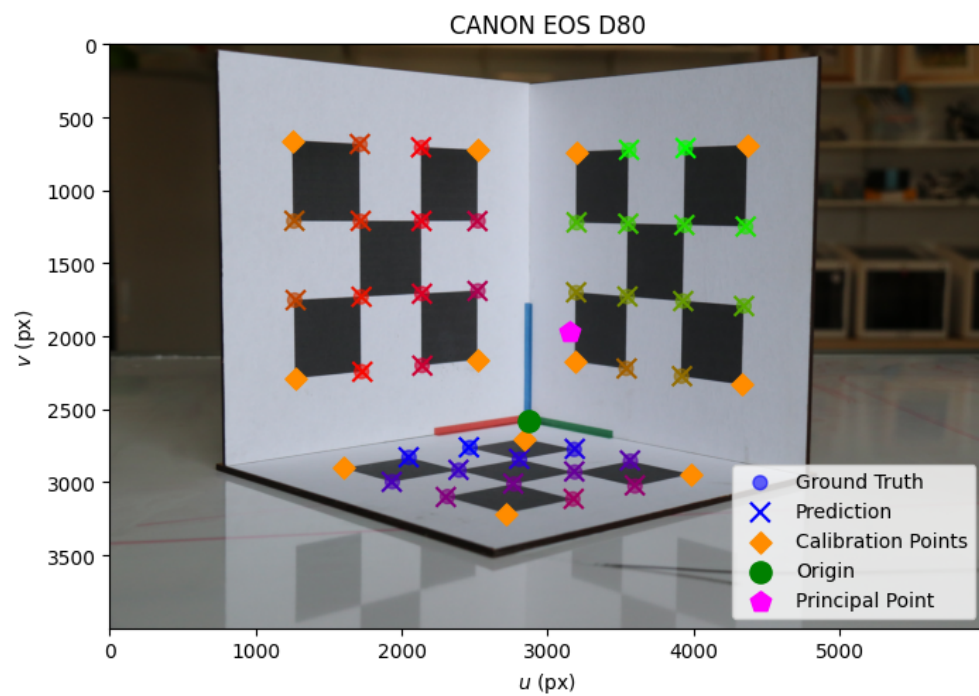
$$\begin{aligned}
r_{31} &= -\sin(\beta) \\
\Rightarrow \beta &= \sin^{-1}(-r_{31}) \quad (6.7)
\end{aligned}$$

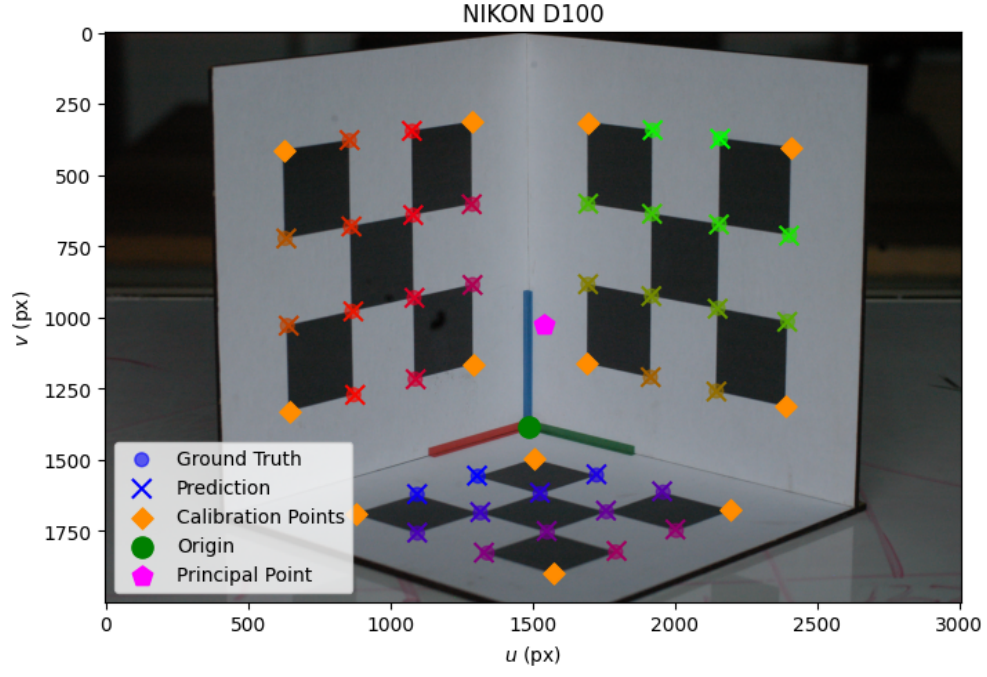
$$\begin{aligned}
r_{32} &= \sin(\alpha) \cos(\beta) \\
\Rightarrow \alpha &= \sin^{-1} \left(\frac{r_{32}}{\cos(\beta)} \right) = \sin^{-1} \left(\frac{r_{32}}{\cos(\sin^{-1}(-r_{31}))} \right) \\
&= \sin^{-1} \left(\frac{r_{32}}{\sqrt{1 - r_{31}^2}} \right) \quad (6.8)
\end{aligned}$$

$$\begin{aligned}
r_{21} &= \cos(\beta) \sin(\gamma) \\
\Rightarrow \gamma &= \sin^{-1} \left(\frac{r_{21}}{\cos(\beta)} \right) = \sin^{-1} \left(\frac{r_{21}}{\cos(\sin^{-1}(-r_{31}))} \right) \\
&= \sin^{-1} \left(\frac{r_{21}}{\sqrt{1 - r_{31}^2}} \right) \quad (6.9)
\end{aligned}$$

7 Experimental Validation

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





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

		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 ¹¹	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 $\pm 1 \text{ mm}$,¹² 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.

¹¹Pixel pitches retrieved from digicamdb.com.

¹²WayneF, “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>.

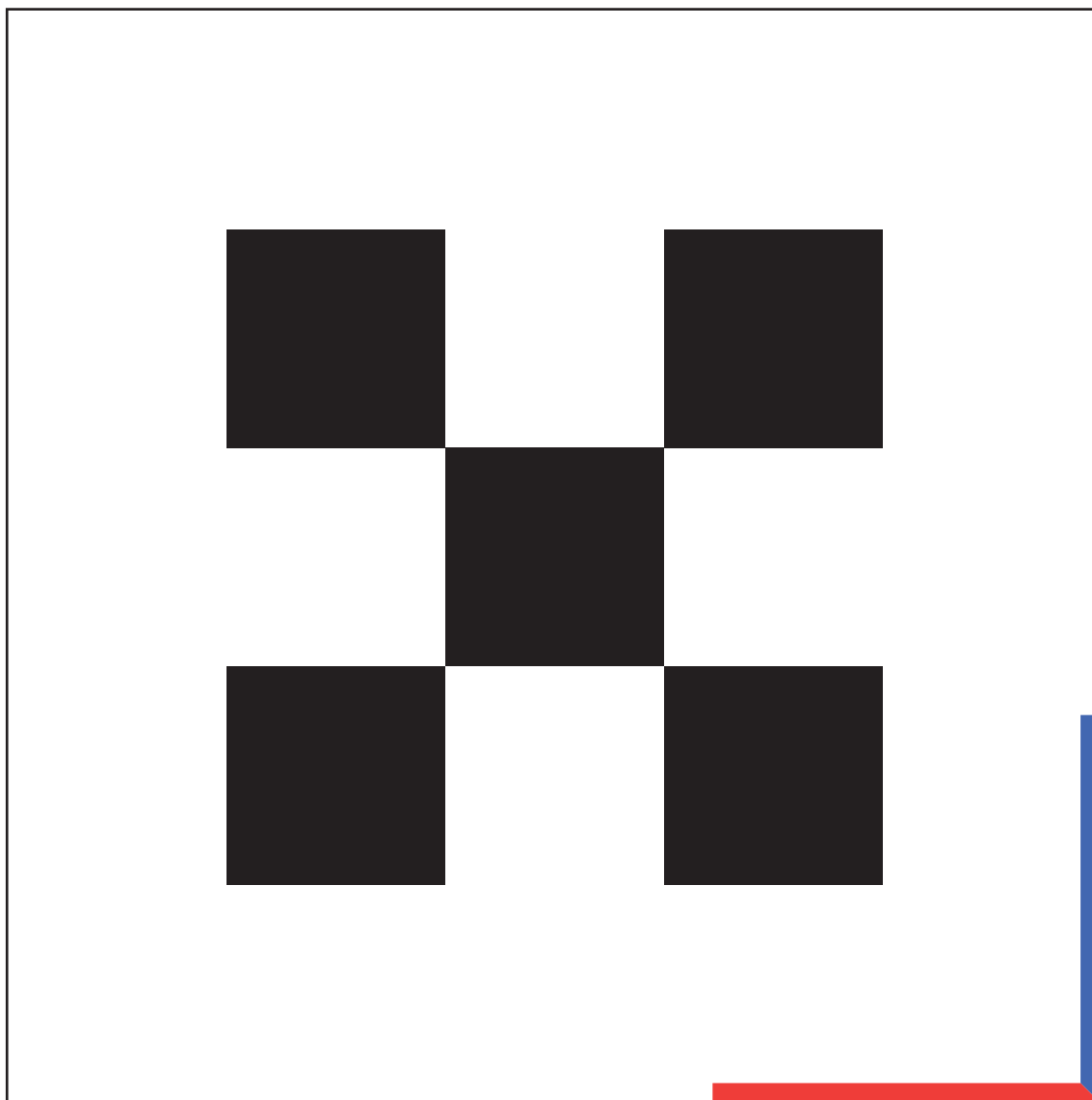
Bibliography

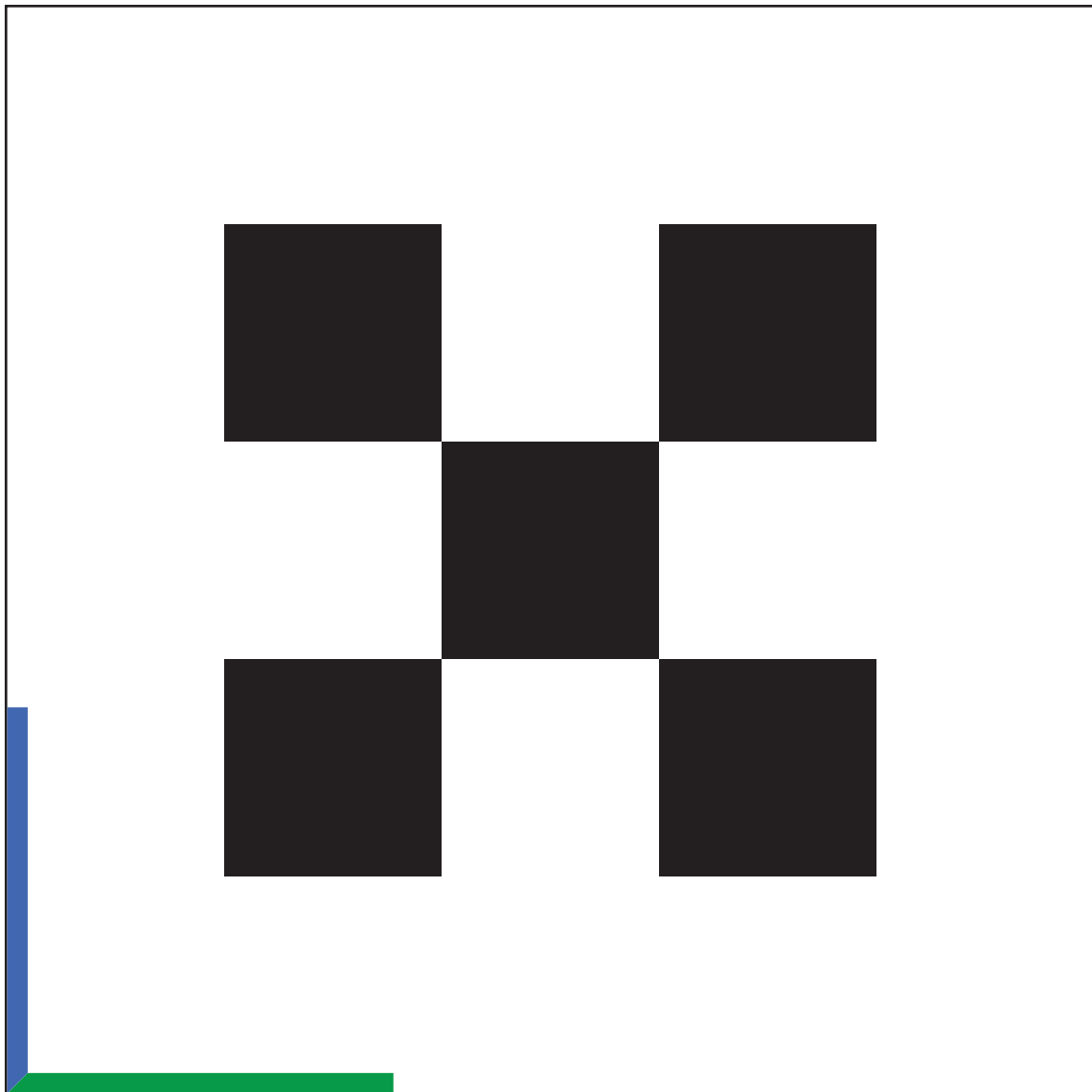
- Albertz, Joerg. "A Look Back; 140 Years of Photogrammetry." *Journal of the American Society for Photogrammetry and Remote Sensing* 73, no. 5 (May 2007): 504–506. Accessed September 9, 2023. <https://www.asprs.org/wp-content/uploads/pers/2007journal/may/lookback.pdf>.
- Bloomenthal, Jules, and Jon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

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

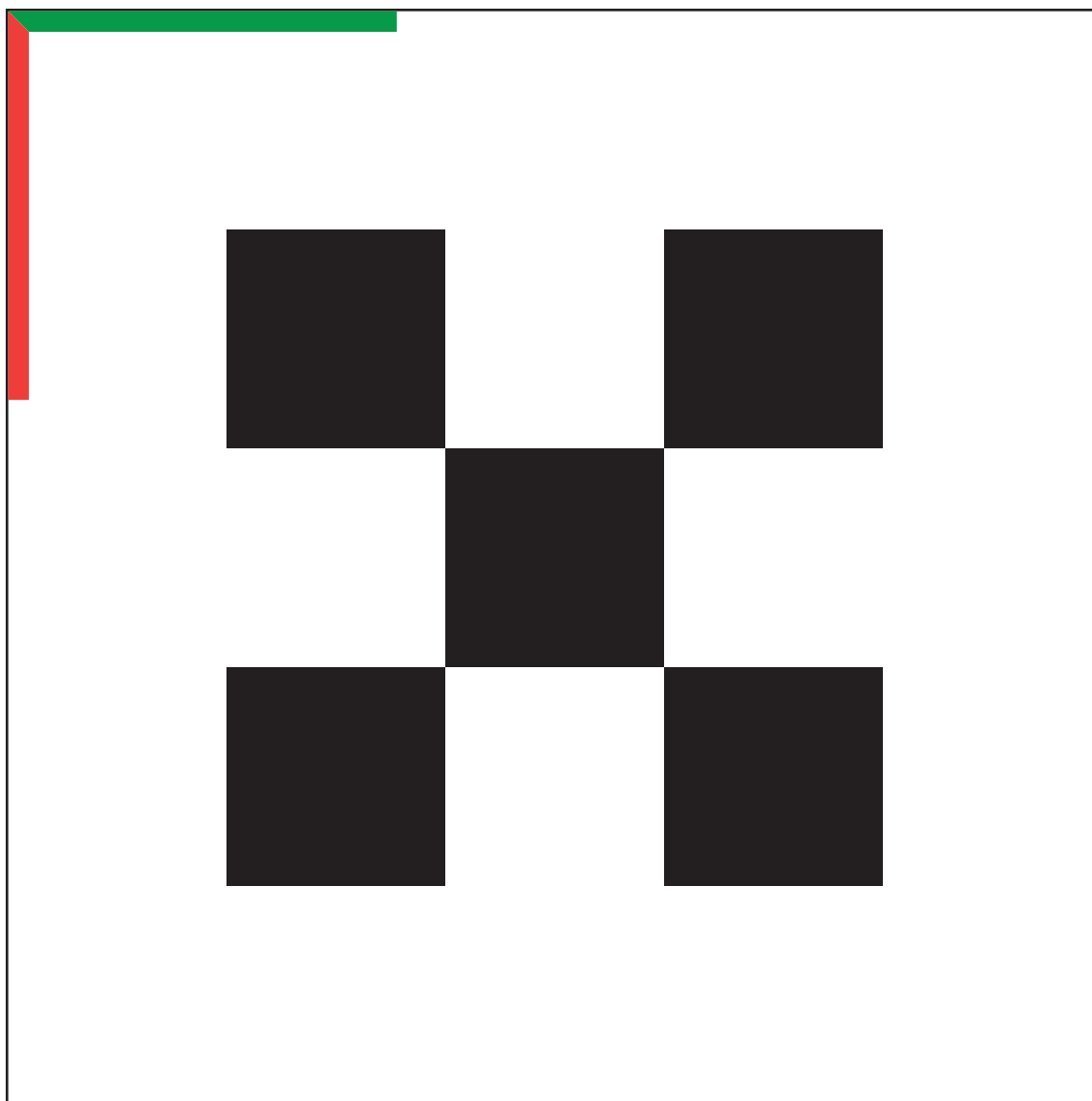
Appendix A Calibration Object Panel Patterns

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

```

run.py

```

1  #!/usr/bin/env python3
2  import os
3  import sys
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from argparse import ArgumentParser, RawDescriptionHelpFormatter
7
8  import calicam
9
10
11 def main():
12     np.set_printoptions(precision=3, suppress=True)
13
14     MAX_HELP_POSITION = 40
15
16     parser = ArgumentParser(
17         prog="calicam",
18         description=(
19             "Generates projection matrix and calculates intrinsic and extrinsic parameters.\n"
20             "CSV inputs are in the format: x,y,z,u,v where 3D point = (x, y, z) and 2D point = (u,v)",
21         ),
22         formatter_class=lambda prog: RawDescriptionHelpFormatter(prog, max_help_position=MAX_HELP_POSITION),
23     )
24
25     parser.add_argument("path", metavar="PATH", help="path to csv file with calibration points")
26     parser.add_argument(
27         "-d", "--data", metavar="DATA_PATH", help="path to csv file with model verification data")
28     parser.add_argument(
29         "-g", "--graph", nargs="?", const="", metavar="BKGD_IMG", help="generate graph")
30     parser.add_argument("-t", "--title", metavar="TITLE", help="title of graph (ignored if `-g` is not
31         ↪ passed)")
32     parser.add_argument("-s", "--show", action="store_true", help="show graph (only necessary if `-o` is
33         ↪ passed)")
34     parser.add_argument("-o", "--out", metavar="GRAPH_PATH", help="graph output location ")
35     parser.add_argument("--noprint", action="store_true", help="don't print output to terminal")
36
37     args = parser.parse_args()
38
39     output = []
40
41     try:
42         # GENERATE MODEL
43         csv_path: str = args.path
44
45         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}
87             ↪ px")
88
89     # OUTPUT
90     if not args.noprint:
91         print(*output, sep="\n")
92
93     # GRAPH
94     image_path: str | None = args.graph
95
96     if image_path is not None:
97         ax: plt.Axes
98         _, ax = plt.subplots(figsize=(8, 10))
99
100         plt.gca().invert_yaxis()
101
102         # image was provided
103         if image_path != "":
104             assert os.path.isfile(image_path), f"{image_path} does not exist."
105             assert args.data, "Path to data csv file must be provide using -d flag to produce graph."
106
107             img = plt.imread(image_path,)
108             ax.imshow(img, cmap='gray')
109             ax.autoscale(False)

```

```

110     # graph data points and model points only if -d flag is specified
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/parser.py

```

1  import csv
2
3  from .vecs import *
4
5
6  def parse_data_from_csv(path: str) -> tuple[list[Vec3f], list[Vec2f]]:
7      """
8      Parses a csv and returns a list of 3D scene points and their corresponding
9      2D image mappings.
10     CSV format: x,y,z,u,v
11     where 3D point = (x, y, z) and 2D point (u,v)
12     """
13     with open(path, "r") as f:
14         reader = csv.reader(f, delimiter=",")
15
16         world_coords = []
17         image_coords = []
18         for lno, line in enumerate(reader, start=1):
19             assert len(line) == 5, f"Data on line {lno} in {path} is invalid."
20
21             x, y, z, u, v = (float(s) for s in line)
22
23             world_coords.append((x, y, z))
24             image_coords.append((u, v))
25
26     return world_coords, image_coords

```

calicam/projection.py

```

1  import numpy as np
2  import scipy.sparse.linalg
3  from nptyping import Shape, Double
4
5  from .vecs import *
6
7  ProjMatrix = np.ndarray[Shape["3, 4"], Double]
8
9
10 def generate_estimation_matrix(world_coords: list[Vec3f], image_coords: list[Vec2f]) -> np.ndarray:
11     """
12     Generates an estimation matrix from list of 3D world coords and
13     their corresponding pixel coord mappings
14     """
15     rows = []
16     for (x, y, z), (u, v) in zip(world_coords, image_coords):
17         rows.append([x, y, z, 1.0, 0.0, 0.0, 0.0, 0.0, -u * x, -u * y, -u * z, -u])
18         rows.append([0.0, 0.0, 0.0, 0.0, x, y, z, 1.0, -v * x, -v * y, -v * z, -v])
19     return np.array(rows)
20
21
22 def generate_proj_matrix(world_coords: list[Vec3f], image_coords: list[Vec2f]) -> tuple[ProjMatrix, float]:
23     """
24     Takes 3D calibration points their corresponding pixel coord mappings and
25     returns the projection matrix as a 3x4 matrix
26     """
27     assert len(world_coords) == len(image_coords), \
28         f"The number of world coordinates ({world_coords}) and image coordinates ({image_coords}) do not  

29         ↪ match."
30
31     assert len(world_coords) >= 6, \

```

```

31         f"Need at least 6 calibration points, but only {len(world_coords)} were provided."
32
33     G = generate_estimation_matrix(world_coords, image_coords)
34     M = G.T @ G
35
36     eigval, p = scipy.sparse.linalg.eigs(M, k=1, which="SM") # solve for minimum p using eigenvalue problem
37     proj_matrix = p.real.reshape(3, 4) # take only real part of p and convert into 3x4 matrix
38
39     return proj_matrix, eigval
40
41
42 def project_point(projection_matrix: ProjMatrix, world_coords: Vec3f) -> Vec2f:
43     """
44     Calculate pixel coordinate from 3D world coord using projection matrix
45     """
46     return to_inhomogenous(projection_matrix @ to_homogenous(world_coords)) # turn into inhomogenous coords

```

calicam/extract.py

```

1  import numpy as np
2  import scipy.linalg
3  from math import sqrt
4  from nptyping import Shape, Double
5
6  from .projection import generate_proj_matrix, ProjMatrix
7  from .vecs import *
8
9  CalMatrix = np.ndarray[Shape["3, 3"], Double]
10 RotMatrix = np.ndarray[Shape["3, 3"], Double]
11
12
13 def calibrate_camera(world_coords: list[Vec3f],
14                     image_coords: list[Vec2f]) -> tuple[ProjMatrix, CalMatrix, RotMatrix, Vec3f]:
15     """
16     Decomposes the projection matrix into the calibration matrix,
17     rotation matrix, and translation matrix.
18     """
19     proj_matrix, _ = generate_proj_matrix(world_coords, image_coords)
20
21     K, R = scipy.linalg.rq(proj_matrix[:, :3]) # rq decomposition
22
23     # enforce positive diagonal on K
24     D = np.diag(np.sign(np.diag(K)))
25     K = K @ D
26     R = D @ R
27
28     # scale projection matrix and calibration matrix to reflect real world scaling
29     scale_factor = 1 / K[2][2]
30     proj_matrix *= scale_factor
31     K *= scale_factor
32
33     # extract translation vector from P
34     t = tuple(-np.linalg.inv(proj_matrix[:, :3]) @ proj_matrix[:, 3])
35
36     return proj_matrix, K, R, t
37
38
39 def extract_intrinsics(K: CalMatrix) -> tuple[Vec2f, Vec2f]:
40     """
41     Extract principle point and focal lengths from calibration matrix
42     """
43     principal_point = (K[0][2], K[1][2])
44     focal_lengths = (K[0][0], K[1][1])

```

```

45     return principal_point, focal_lengths
46
47
48 def extract_orientation_zyx(R: RotMatrix) -> Vec3f:
49     """
50     Extract tait-bryan angles (zyx) from rotation matrix
51     """
52     return (
53         np.degrees(np.arcsin(R[2][1] / sqrt(1 - (R[2][0])**2))), # alpha (x rotation)
54         np.degrees(np.arcsin(-R[2][0])), # beta (y rotation)
55         np.degrees(np.arcsin(R[1][0] / sqrt(1 - (R[2][0])**2))), # gamma (z rotation)
56     )

```

calicam/vecs.py

```

1  from math import sqrt
2
3  Vecf = tuple[float, ...]
4
5  Vec2f = tuple[float, float]
6  Vec3f = tuple[float, float, float]
7
8
9  def to_homogenous(vec: Vecf) -> Vecf:
10     return (*vec, 1.0)
11
12
13  def to_inhomogenous(vec: Vecf) -> Vecf:
14     return tuple(map(lambda v_i: v_i / vec[-1], vec[:-1]))
15
16
17  def euclidean(a: Vecf, b: Vecf) -> float:
18     return sqrt(sum((b_i - a_i)**2 for a_i, b_i in zip(a, b)))

```