

IB Math Analysis and Approaches (HL)  
Extended Essay  
May 2024 Session

---

Mathematical Techniques and Applications of  
Camera Calibration

---

**Research Question:** What mathematical techniques can be employed to develop highly accurate camera models, and what are their real-world applications where these models prove valuable?

**Word Count:** 2091 words

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	1
<b>2</b>	<b>Approach</b>	<b>2</b>
2.1	Camera Model . . . . .	2
2.1.1	Pinhole Camera Model . . . . .	3
2.2	Calibration Object . . . . .	4
<b>3</b>	<b>Prerequisites</b>	<b>5</b>
3.1	Notation . . . . .	5
3.2	Homogenous Coordinates . . . . .	5
<b>4</b>	<b>Constructing the Pinhole Camera Model</b>	<b>7</b>
4.1	Nomenclature . . . . .	7
4.2	Intrinsic Parameters . . . . .	8
4.3	Extrinsic Parameters . . . . .	11
4.4	Putting It All Together . . . . .	13
<b>5</b>	<b>Camera Calibration</b>	<b>14</b>
5.1	Overall Strategy . . . . .	14
5.2	Solving for the Projection Matrix . . . . .	14
5.3	Constrained Least Squares Solution . . . . .	15
<b>6</b>	<b>Extracting Parameters</b>	<b>17</b>
6.1	RQ Decomposition . . . . .	17
6.2	Extracting the Translation Vector . . . . .	18
6.3	Extracting Orientation as Angles . . . . .	18
<b>7</b>	<b>Experimental Validation</b>	<b>19</b>
<b>8</b>	<b>Applications</b>	<b>20</b>

<b>Acknowledgements</b>	<b>20</b>
<b>Bibliography</b>	<b>22</b>
<b>Appendix A Calibration Object Details</b>	<b>23</b>
A.1 Panels . . . . .	23
A.2 Grid Pattern . . . . .	23
<b>Appendix B Source Code</b>	<b>27</b>

# 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<sup>1</sup>. 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 applications requiring high accuracy. Combined with the potential for manufacturing defects

---

<sup>1</sup>Albertz, "A Look Back; 140 Years of Photogrammetry," 1.

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 different real-world applications.

## 2 Approach

There are countless different approaches one could take to calibrate a camera,

however they all build upon techniques first described in multiple highly influential papers, most notably Tsai’s “A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses” and Zhang’s “A Flexible New Technique for Camera Calibration”.

### 2.1 Camera Model

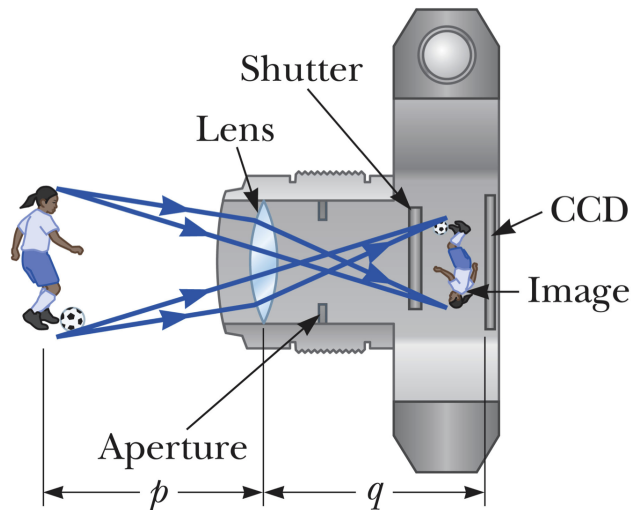
A camera model is a projection model which approximates the function of a camera by describing a mathematical relationship between points in 3D space and its projection onto the sensor grid of the camera. In order to construct such a model, we must first understand the general workings of a camera.

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 the projected image created by the lens and the aperture.



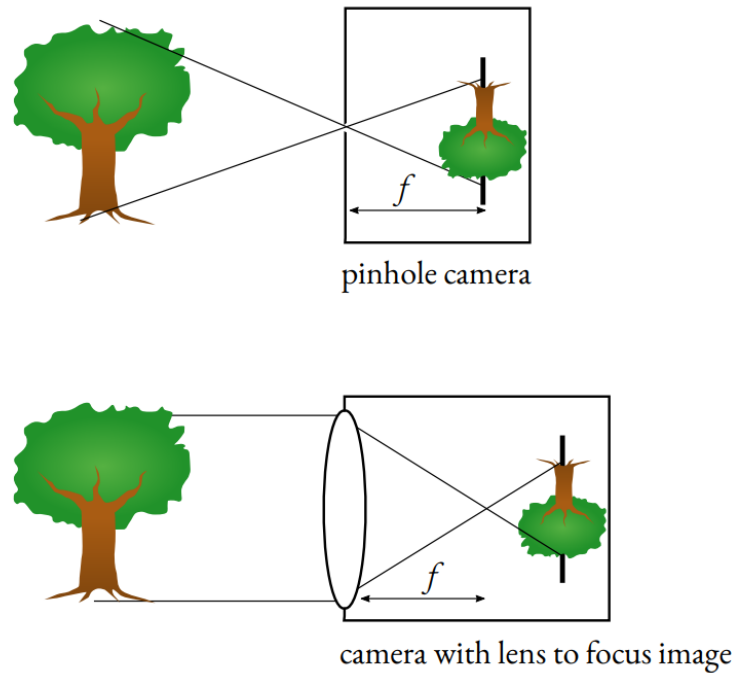
**Figure 2.1:** Lens camera. Adapted from Colton, “Warm-Up Exercise 30.”

However, even this simplified model of a lens camera is still too complicated to describe, as it is impossible to encapsulate the complex behavior of a lens succinctly using one simple mathematical equation. As such, a simpler model needs to be devised by distilling the behavior to its most fundamental and essential dynamics. In doing so, we exclude certain details, but as we will see

### 2.1.1 Pinhole Camera Model

A pinhole camera is a simple camera without a lens. Instead, it 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 a few important assumptions for the purpose of simplification:

1. **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. This allows us to establish a relationship between a 3D point and its 2D projection onto the image plane.



**Figure 2.2:** Difference between a pinhole camera and a lens camera. Adapted from Lê, “Camera Model: Intrinsic Parameters.”

## 2. Infinite depth of field –

Extremely simple model for imaging geometry Doesn’t strictly apply Mathematically convenient acceptable approximation.

While the pinhole camera model is an extremely basic model that does not accurately describe the true behavior of cameras, it is extremely simple. As such, for the sake of simplifying camera models, the pinhole camera model is often used. This model is one of the most basic and frequently employed camera models in the field of camera calibration.

## 2.2 Calibration Object

Calibration techniques can be roughly separated into 3 categories, based on the dimension of the calibration object used<sup>2</sup>:

- 3D –
- 2D –

---

<sup>2</sup>Zhang, “Camera Calibration.”

- 1D –

### 3 Prerequisites

Given the that this paper utilizes mathematical concepts beyond the scope of the IB Mathematics Analysis and Approaches HL curriculum, it is imperative that some notation and important ideas are introduced prior.

#### 3.1 Notation

**Vectors and Matrices.** In this paper, lowercase letters are used to denote vectors, whereas capital letters are used for matrices. Depending on the context, vectors can also be attached with diacritics:

- $\vec{v}$  – a letter with an arrow above it denotes a positional vector or translational vector dealing with the transformation of points in space.
- $\tilde{v}$  – a letter with a tilde above it denotes a vector represented in homogenous coordinates. This idea is explained in section 3.2.
- $v$  – when explicitly stated, a letter without diacritics can also denote a vector if it does not fall into the categories listed above.

**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:

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

#### 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.

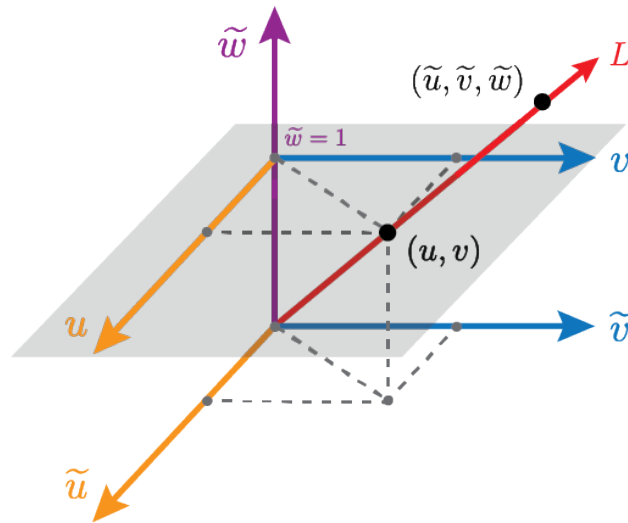
Homogenous coordinates forms the basis of projective geometry, because it unifies the treatment of common graphical transformations such as rotation and translations<sup>3</sup>.

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

When

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

$$\begin{bmatrix} x \\ y \end{bmatrix} \quad (3.1)$$



**Figure 3.1:** Homogenous coordinate system.

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

<sup>3</sup>Bloomenthal and Rokne, “Homogeneous Coordinates,” 1.

## 4 Constructing the Pinhole Camera Model

### 4.1 Nomenclature

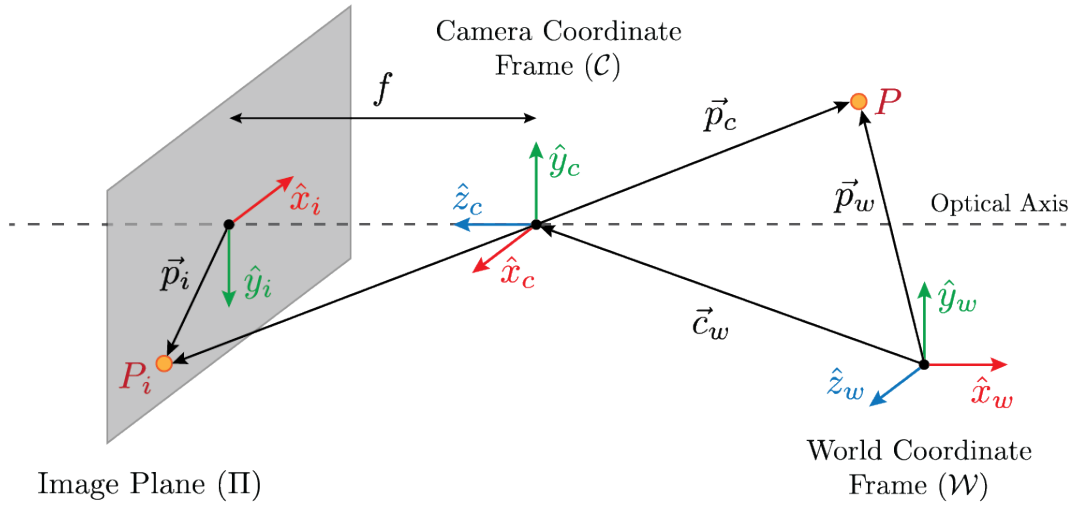


Figure 4.1: Pinhole camera model.

For our camera model, we will introduce 4 different coordinate systems:

- **The World Coordinate Frame  $\mathcal{W}$ .** Points are denoted as  $(x_w, y_w, z_w)$
- **The Camera Coordinate Frame  $\mathcal{C}$ .** Points are denoted as  $(x_c, y_c, z_c)$ .
- **The Image Plane  $\Pi$ .** Points are denoted as  $(x_i, y_i)$ .
- **The Sensor Grid.** Points are denoted as  $(u, v)$ .

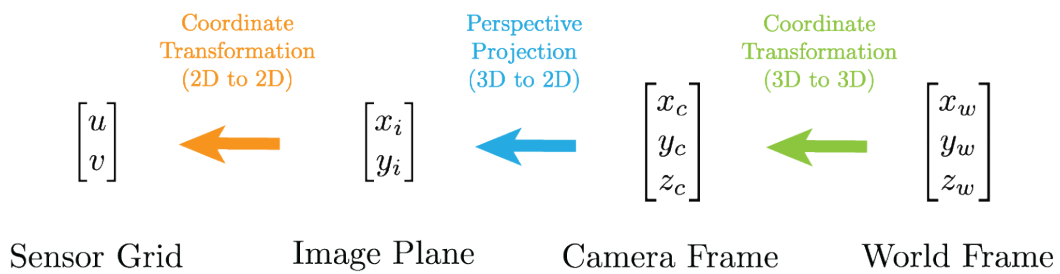


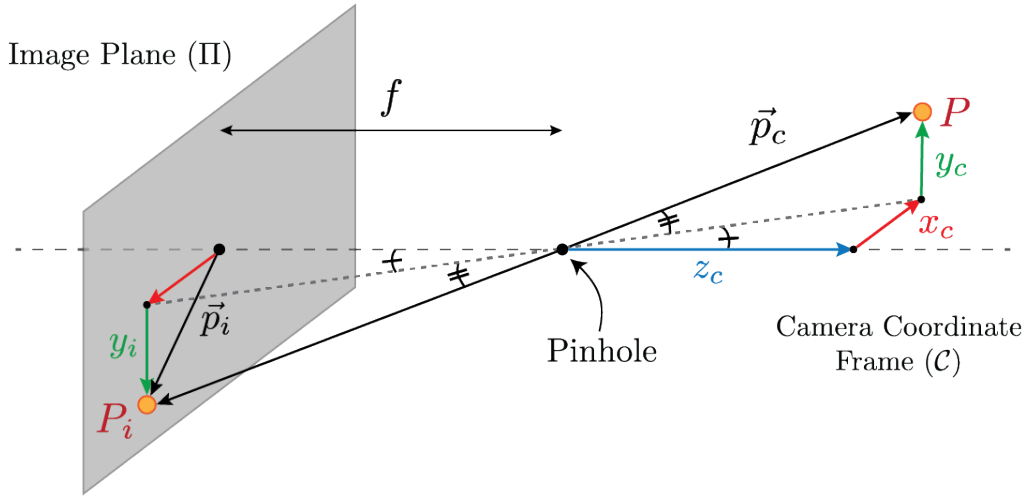
Figure 4.2: Coordinate remappings.

## 4.2 Intrinsic Parameters

First, we will focus on the projection of points in the 3D space onto the image plane. The goal is to construct a calibration matrix,  $K$ , which relates the position of the point  $P$  to its projection on the image plane. This can be expressed mathematically as follows:

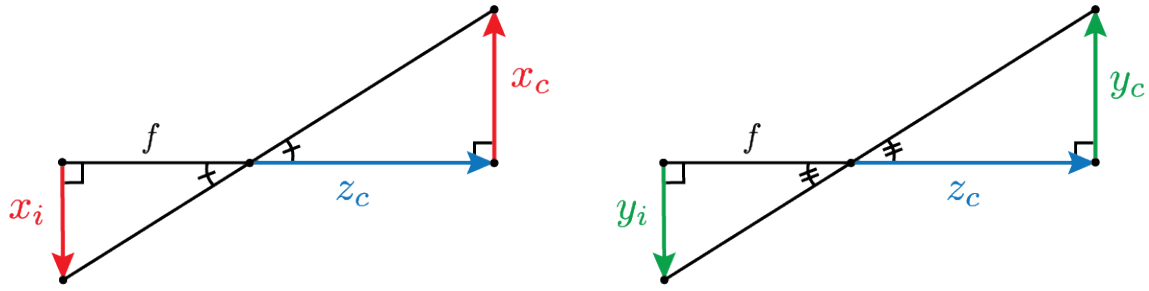
$$\tilde{p}_i = K\tilde{p}_c \quad (4.1)$$

where  $\vec{p}_i$  and  $\vec{p}_c$  represents the position of  $P$  in the image plane  $\Pi$  and the camera frame  $\mathcal{C}$  respectively.



**Figure 4.3:** Perspective projection of the point  $P$  onto the image plane  $\Pi$ .

When a straight line is drawn from  $P$  to its projection  $P_i$  through the aperture, it intersects the optical axis. Deconstructing this intersection in the  $x$  and  $y$  direction, pairs of similar triangles are formed on the  $x$  and  $y$  plane. This is visualized in Figure 4.4.

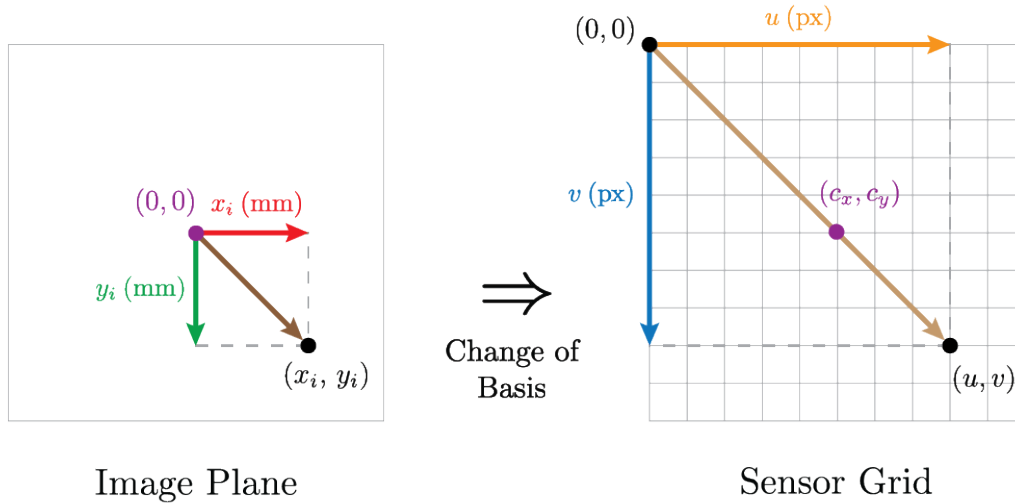


**Figure 4.4:** Similar triangles formed by perspective projection, which relate  $x_i$  to  $x_c$  and  $y_i$  to  $y_c$

$$\frac{x_i}{f} = \frac{x_c}{z_c} \implies x_i = f \frac{x_c}{z_c} \quad (4.2a)$$

$$\frac{y_i}{f} = \frac{y_c}{z_c} \implies y_i = f \frac{y_c}{z_c} \quad (4.2b)$$

We can then relate the coordinates of the projection,  $(x_i, y_i)$ , which are in real-world units, to its position  $(u, v)$  in pixels.



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

Let  $m_x$  and  $m_y$  represent the pixel density of the image sensor in the  $x$  and  $y$  axes of the image sensor plane respectively.

$$u = m_x x_i + c_x$$

$$v = m_y y_i + c_y$$

Replacing  $x_i$  and  $y_i$  for the result we obtained from 4.2a and 4.2b, we get:

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

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

Since  $m_x$ ,  $m_y$ , and  $f$  are all unknowns, we can combine the products  $m_x f$  and  $m_y f$  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.

$$u = f_x \frac{x_c}{z_c} + c_x \tag{4.3a}$$

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

$$\begin{bmatrix} u \\ v \end{bmatrix} \sim \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c c_x \\ f_y y_c + z_c c_y \\ z_c \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{4.4}$$

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Note that  $K$  that 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 intrinsic matrix from the projection matrix in section 5.

As such, we can express  $M_{int}$  as  $[K \mid 0]$ .

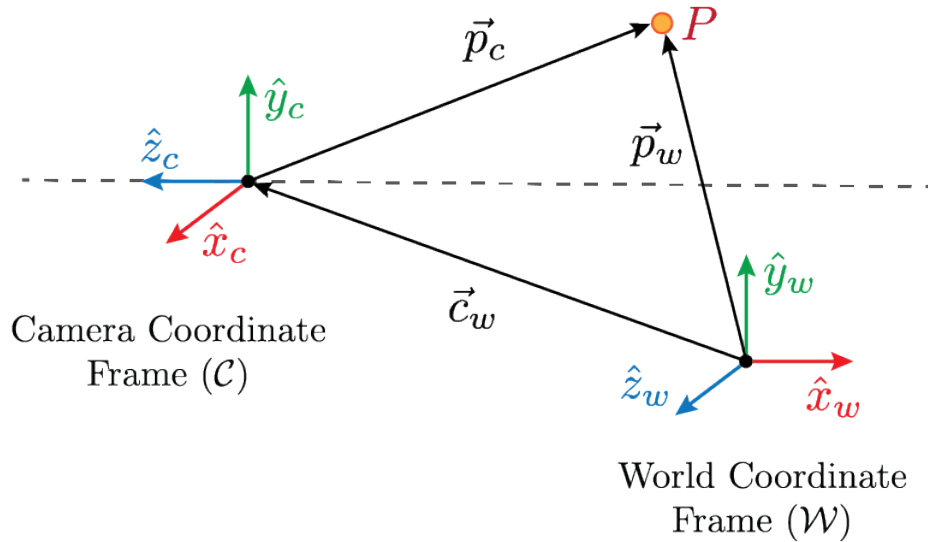
$$M_{int} = [K \mid 0] \quad (4.6)$$

### 4.3 Extrinsic Parameters

Next, we will focus on finding the position

Now, we would like to find the extrinsic matrix,  $M_{ext}$ , which relates the positional vector  $\vec{p}_w$  of point  $P$  in the world coordinate frame, to its positional vector  $\vec{p}_c$  in the camera coordinate frame. Similar to what we did in section 4.2, we can express this in homogenous coordinates as follows:

$$\tilde{p}_c = M_{ext} \tilde{p}_w \quad (4.7)$$



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

For the extrinsic parameters of the camera, we have the position  $\vec{c}_w$  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.

$$\vec{p}_c = R(\vec{p}_w - \vec{c}_w) \quad (4.9a)$$

$$= R\vec{p}_w - R\vec{c}_w \quad (4.9b)$$

$$\vec{p}_c = R\vec{p}_w + \vec{t} \quad (4.10)$$

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

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M_{ext}} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (4.12)$$

$$M_{ext} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \quad (4.13)$$

#### 4.4 Putting It All Together

When we combine the equations  $\tilde{p}_c = M_{ext} \tilde{p}_w$  (eq. 4.7) and  $\tilde{p}_i = M_{int} \tilde{p}_c$  (eq. 4.1), we obtain

$$\tilde{p}_i = M_{int} M_{ext} \tilde{p}_w \quad (4.14)$$

To simplify our camera model, we can define a new matrix,  $P \in \mathbb{R}^{3 \times 4}$ , which is equal to the product  $M_{int} M_{ext}$ . Since  $M_{ext}$  is a  $4 \times 4$  matrix and  $M_{int}$  is a  $3 \times 4$  matrix, their matrix product produces a  $3 \times 4$  matrix.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \equiv \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{[R | \vec{t}]} \quad (4.15)$$

Replacing  $P$  for  $M_{int} M_{ext}$  in equation 4.14, we obtain

$$\tilde{p}_i = P \tilde{p}_w \quad (4.16)$$

The implications of this equation is very important, as it means that we can project the  $n$ th point  $[x_w^{(n)}, y_w^{(n)}, z_w^{(n)}]^\top$  in the world coordinate frame  $\mathcal{W}$  to its pixel coordinates  $[u_n, v_n]^\top$  on the image plane  $\Pi$  simply by using the projection matrix. But now, we need to figure out a way to solve for the project matrix.

Given that we have equation 4.16 which relates



When expressing the pixel coordinate in homogenous coordinates, equation 4.16 becomes

$$\begin{bmatrix} u_n \\ v_n \end{bmatrix} \sim \begin{bmatrix} \tilde{u}_n \\ \tilde{v}_n \\ \tilde{w}_n \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(n)} \\ y_w^{(n)} \\ z_w^{(n)} \\ 1 \end{bmatrix} \quad (4.17)$$

## 5 Camera Calibration

### 5.1 Overall Strategy

### 5.2 Solving for the Projection Matrix

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

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

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

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

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

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

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



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 5.3, we can replace  $\|Gp\|^2$  with  $p^\top A^\top Gp$  and  $\|p\|^2$  for  $p^\top p$  to obtain

$$\underset{p}{\text{minimize}} \quad (p^\top G^\top Gp) \quad \text{subject to} \quad p^\top p = 1 \quad (5.4)$$

The Lagrangian<sup>4</sup> of equation 5.4 is

$$\mathcal{L}(p, \lambda) = p^\top G^\top Gp - \lambda (p^\top p - 1) \quad (5.5)$$

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  $\lambda$  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  $\partial$  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:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p} &= \frac{\partial}{\partial p} [p^\top G^\top Gp - \lambda (p^\top p - 1)] \stackrel{\text{set}}{=} 0 \\ &\Rightarrow 2G^\top Gp - 2\lambda p = 0 \\ &\Rightarrow G^\top Gp = \lambda p \end{aligned} \quad (5.6)$$

which is an eigenvalue problem for  $G^\top G$ . Potential solutions for  $p$  are eigenvectors that satisfy equation 5.6,<sup>5</sup> with  $\lambda \in \mathbb{R}$  as the eigenvalue. Since 5.4 is a minimization problem, the

<sup>4</sup>Ghojogh, Karray, and Crowley, "Eigenvalue and Generalized Eigenvalue Problems," 2.

<sup>5</sup>Nayar, *Linear Camera Model*.

minimized eigenvector  $p$  is the one which has the smallest eigenvalue  $\lambda$ .<sup>6</sup>

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

$$\begin{aligned} P &= K [R | \vec{t}] \\ &= K [R | -R\vec{c}_w] \\ &= [KR | -KR\vec{c}_w] \end{aligned} \tag{6.1}$$

$$P = [Q | -Q\vec{c}_w] \tag{6.2}$$

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

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

---

<sup>6</sup>Ghojogh, Karray, and Crowley, “Eigenvalue and Generalized Eigenvalue Problems,” 2.

## 6.2 Extracting the Translation Vector

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

## 6.3 Extracting Orientation as Angles

When constructing the extrinsic matrix in section 4.3, 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

		CANON D80		IPHONE X		NIKON D100	
		Spreaded Out	Concentrated	Spreaded Out	Concentrated	Spreaded Out	Concentrated
Focal Lengths	$f_x$	8404.1 px	8305.9 px	3281.5 px	3125.7 px	8144.4 px	7716.4 px
	$f_y$	8387.9 px	8338.1 px	3279.9 px	3137.8 px	8142.6 px	7755.5 px
Principal Point	$c_x$	3151.6 px	3400.0 px	2043.0 px	2031.4 px	1541.8 px	1851.5 px
	$c_y$	1972.8 px	2075.8 px	1453.1 px	1467.4 px	1027.9 px	1205.4 px
Tait-Bryan Angles	$\alpha$	$-81.86^\circ$	$-80.32^\circ$	$-60.21^\circ$	$-60.02^\circ$	$-70.83^\circ$	$-67.83^\circ$
	$\beta$	$44.27^\circ$	$46.00^\circ$	$38.72^\circ$	$38.28^\circ$	$46.44^\circ$	$48.43^\circ$
	$\gamma$	$4.97^\circ$	$5.98^\circ$	$21.64^\circ$	$21.81^\circ$	$13.89^\circ$	$16.04^\circ$
Translation	$t_x$	494.8 mm	492.9 mm	329.0 mm	317.4 mm	840.3 mm	801.1 mm
	$t_y$	537.6 mm	533.5 mm	321.4 mm	310.2 mm	766.0 mm	726.7 mm
	$t_z$	128.3 mm	130.5 mm	208.6 mm	202.1 mm	317.2 mm	306.8 mm
Reproj. Errors	$\mu_{max}$	11.08 px	17.21 px	5.58 px	19.48 px	11.70 px	14.13 px
	$\mu_{avg}$	3.56 px	7.36 px	2.55 px	4.86 px	2.81 px	5.04 px

**Table 7.1:** Intrinsic and Extrinsic Parameters calculated by `calicam`.

**Figure 7.1:** Photograph 1. The photo editing software *GIMP* was used for edge detection, and the coordinates of the calibration points were selected manually.

**Figure 7.2:** Graph produced by `Matplotlib` which displays the results of the trial

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

## 8 Applications

### 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.



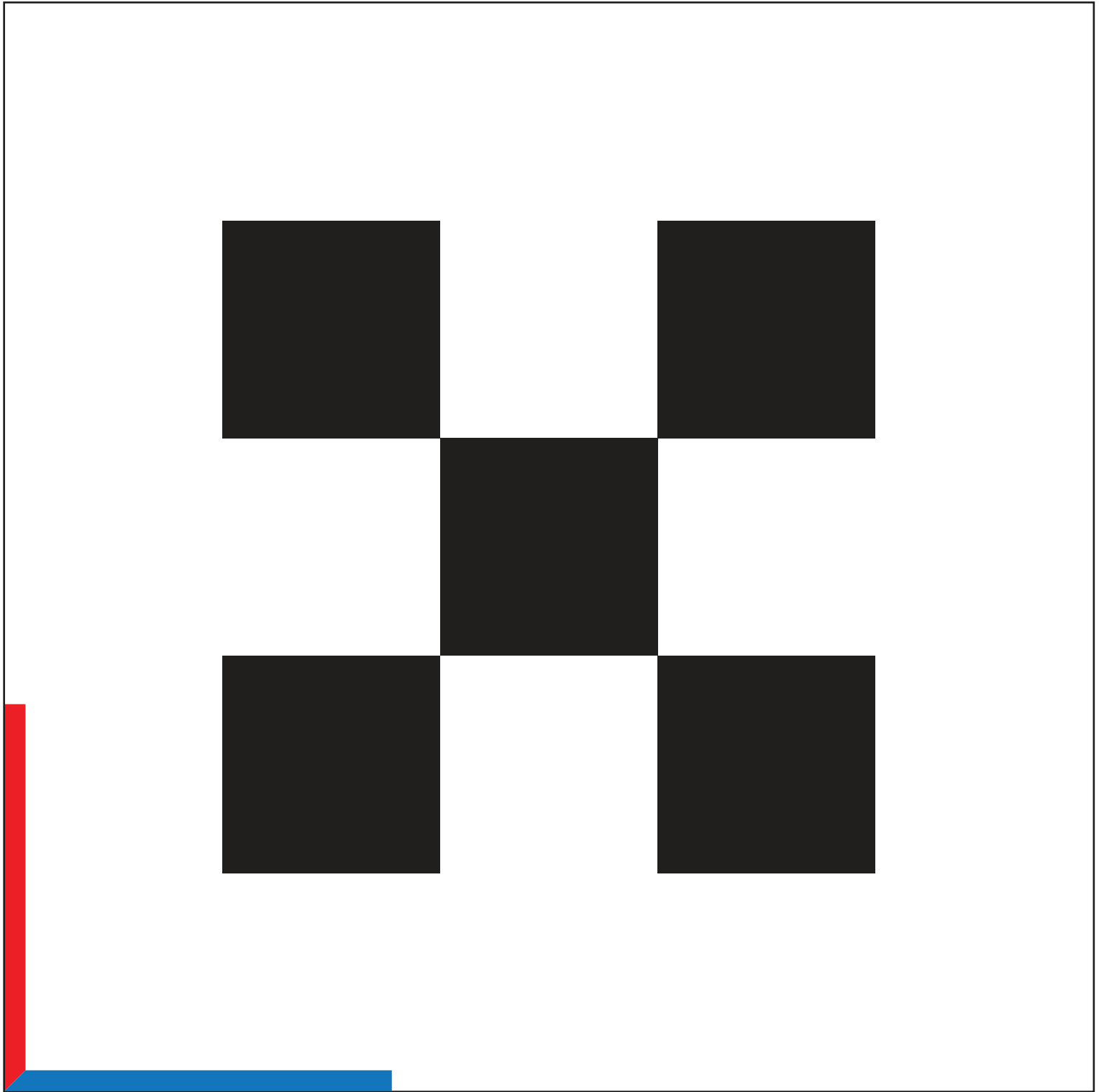
## Bibliography

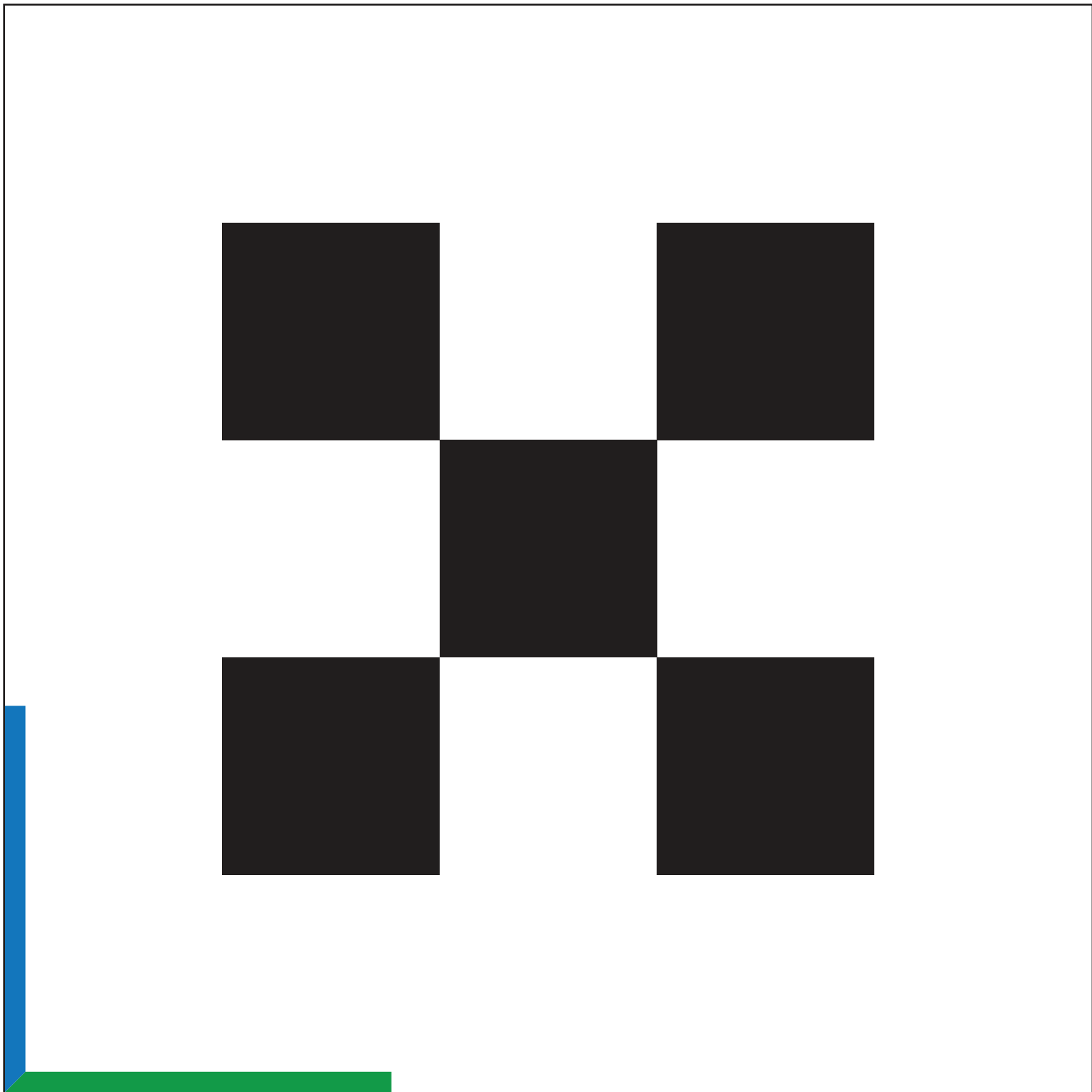
- Albertz, Joerg. “A Look Back; 140 Years of Photogrammetry.” *Journal of the American Society for Photogrammetry and Remote Sensing* 73, no. 5 (May 2007): 504–506. Accessed September 9, 2023. <https://www.asprs.org/wp-content/uploads/pers/2007journal/may/lookback.pdf>.
- Bloomenthal, Jules, and Jon 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>.
- Zhang, Zhengyou. “Camera Calibration,” May 2007. Accessed October 10, 2023. <https://people.cs.rutgers.edu/elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf>.

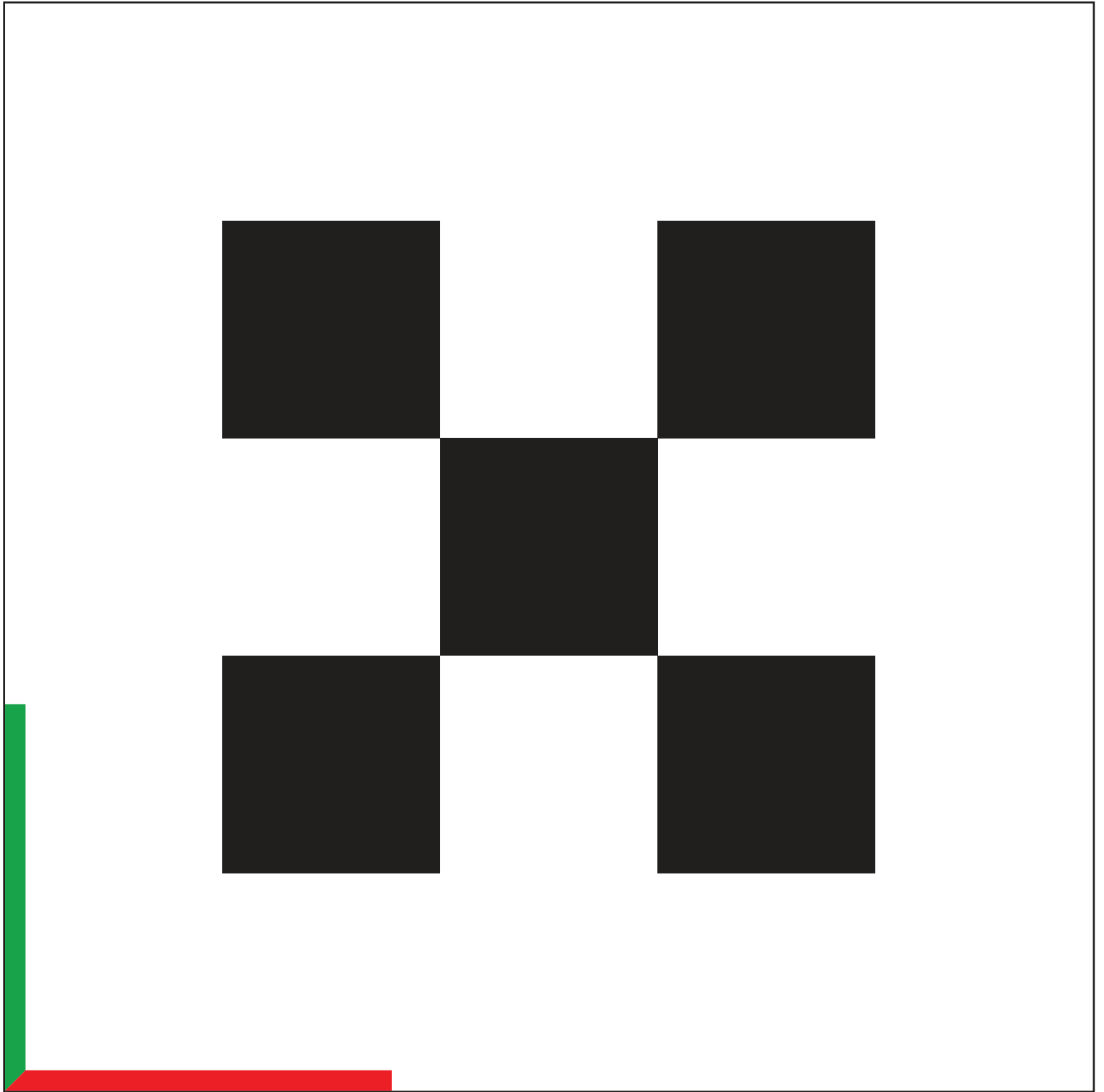
## Appendix A Calibration Object Details

### A.1 Panels

### A.2 Grid Pattern







## Appendix B Source Code

### Project 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             formatter_class=lambda prog: RawDescriptionHelpFormatter(prog, max_help_position=MAX_HELP_POSITION),
22         )
23
24     parser.add_argument("path", metavar="PATH", help="path to csv file with calibration points")
25     parser.add_argument(
26         "-d", "--data", metavar="DATA_PATH", help="path to csv file with model verification data")
27     parser.add_argument(
28         "-g", "--graph", nargs="?", const="", metavar="BKGD_IMG", help="generate graph")
29     parser.add_argument("-t", "--title", metavar="TITLE", help="title of graph (ignored if `-g` is not
    ↪ passed)")
```

```

30     parser.add_argument("-s", "--show", action="store_true", help="show graph (only necessary if `-o` is
    ↪ passed)")
31     parser.add_argument("-o", "--out", metavar="GRAPH_PATH", help="graph output location ")
32     parser.add_argument("--noprint", action="store_true", help="don't print output to terminal")
33
34     args = parser.parse_args()
35
36     output = []
37
38     try:
39         # GENERATE MODEL
40         csv_path: str = args.path
41
42         cali_world_coords, cali_image_coords = calicam.parse_data_from_csv(csv_path)
43         proj_matrix, cali_matrix, rot_matrix, (tx, ty, tz) = calicam.calibrate_camera(
44             cali_world_coords, cali_image_coords)
45
46         (cx, cy), (fx, fy) = calicam.extract_intrinsics(cali_matrix)
47         a, b, g = calicam.extract_orientation_zyx(rot_matrix)
48
49         log_path: str = args.path
50
51         output.append("\n" + "\n\n".join((
52             f"Projection Matrix: \n{proj_matrix}",
53             f"Calibration Matrix: \n{cali_matrix}",
54             f"Rotation Matrix: \n{rot_matrix}",
55             f"Focal Lengths: \n\tf_x = {fx:.2f} px \n\tf_y = {fy:.2f} px",
56             f"Principal Point: \n\tc_x = {cx:.2f} px \n\tc_y = {cy:.2f} px",
57             f"Translation: \n\tt_x = {tx:.2f} \n\tt_y = {ty:.2f} \n\tt_z = {tz:.2f}",
58             f"Orientation: \n\t\u03B1 = {a:.2f}° \n\t\u03B2 = {b:.2f}° \n\t\u03B3 = {g:.2f}°",
59         )))
60
61         # MODEL VALIDATION
62         data_path: str | None = args.data
63
64         if data_path is not None:
65             assert os.path.isfile(data_path), f"{data_path} does not exist."
66             assert data_path.endswith(".csv"), f'{data_path} does not end with the extension ".csv".'
67
68             data_world_coords, data_image_coords = calicam.parse_data_from_csv(data_path)
69
70             predicted_coords = [
71                 calicam.project_point(proj_matrix, world_coord) for world_coord in data_world_coords
72             ]
73             reproj_errs = [

```

```

74         calicam.euclidean(actual_coord, reproj_coord)
75     for actual_coord, reproj_coord in zip(data_image_coords, predicted_coords)
76 ]
77
78     max_err = max(reproj_errs)
79     avg_err = sum(reproj_errs) / len(reproj_errs)
80
81     output.append(
82         f"\nReprojection Errors: \n\t\u03BC_max = {max_err:.3f} px \n\t\u03BC_avg = {avg_err:.3f}
83         ↪ px")
84
85     # OUTPUT
86     if not args.noprint:
87         print(*output, sep="\n")
88
89     # GRAPH
90     image_path: str | None = args.graph
91
92     if image_path is not None:
93         ax: plt.Axes
94         _, ax = plt.subplots(figsize=(8, 10))
95
96         plt.gca().invert_yaxis()
97
98         # image was provided
99         if image_path != "":
100             assert os.path.isfile(image_path), f"{image_path} does not exist."
101             assert args.data, "Path to data csv file must be provide using -d flag to produce graph."
102
103             img = plt.imread(image_path,)
104             ax.imshow(img, cmap='gray')
105             ax.autoscale(False)
106
107             # graph data points and model points only if -d flag is specified
108             if data_path is not None:
109                 cmap = plt.cm.brg
110                 discrete_cmap = list(cmap(np.linspace(0, 1, len(data_image_coords))))
111
112                 # data points
113                 ax.scatter(
114                     *zip(*data_image_coords),
115                     label="Ground Truth",
116                     s=50,
117                     color=discrete_cmap,
118                     marker="o",

```



```

118         alpha=0.6,
119     )
120
121     # predicted points
122     ax.scatter(
123         *zip(*predicted_coords),
124         label="Prediction",
125         s=100,
126         color=discrete_cmap,
127         marker="x",
128     )
129
130     # calibration points
131     ax.scatter(
132         *zip(*cali_image_coords),
133         label="Calibration Points",
134         s=60,
135         marker="D",
136         color="darkorange",
137     )
138
139     # principle point
140     ax.scatter(cx, cy, label="Principal Point", s=120, marker="p", color="magenta")
141
142     graph_title: str = args.title or image_path
143
144     plt.gca().update({"title": graph_title, "xlabel": "$u$ (px)", "ylabel": "$v$ (px)"})
145     plt.legend()
146
147     out_path: str | None = args.out
148
149     if out_path:
150         plt.savefig(out_path)
151
152     # show graph if -s flag was specified or if a save location was not specified
153     if args.show or not out_path:
154         plt.show()
155
156     except AssertionError as e:
157         parser.error(str(e)) # pass error to argparse
158
159     except KeyboardInterrupt:
160         print(f"\nKeyboardInterrupt")
161         sys.exit(1)
162

```

```
163     sys.exit(0)
164
165
166 if __name__ == "__main__":
167     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, -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, \
32         f"Need at least 6 calibration points, but only {len(world_coords)} were provided."
33
34     G = generate_estimation_matrix(world_coords, image_coords)
35     M = G.T @ G
36
37     eigval, p = scipy.sparse.linalg.eigs(M, k=1, which="SM") # solve for minimum p using eigenvalue problem
38     proj_matrix = p.real.reshape(3, 4) # take only real part of p and convert into 3x4 matrix
39
40     return proj_matrix, eigval
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     im_point = projection_matrix @ to_homogenous(world_coords)
47     return to_inhomogenous(im_point) # 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     focal_lengths = (K[0][0], K[1][1])
44     principal_point = (K[0][2], K[1][2])
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)))

```

