Computation of the Camera Matrix

Bryan Lee

December 13, 2024

1 Introduction

In computer vision, the camera matrix P is a $3x4$ matrix that defines the mathematical transformation mapping of 3-D points in world space into their corresponding 2-D coordinates on the camera's image plane. Essentially, it models how a camera captures a scene by incorporating both the intrinsic camera parameters (such as focal length, principle point, skew, and distortion scale) and extrinsic parameters (position and orientation) required for this transformation. For our computations, we will utilize the Gold Standard Algorithm and the pinhole camera model. In this model, a 3-D point in the world is projected onto a 2-D image plane through a single point, the pinhole. This projection establishes a mapping between 3-D world coordinates and 2-D image coordinates, determined by the relative position of the point in relation to the camera's optical center.

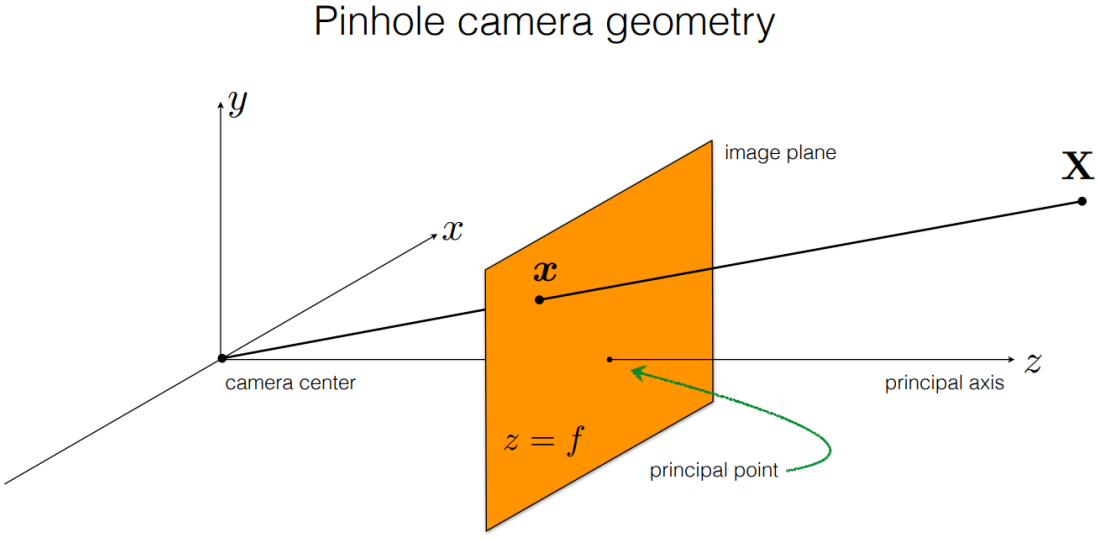

Figure 1: The pinhole camera geometry that will help us compute the camera matrix P.

2 Preliminaries

A standard camera matrix P takes the form:

$$x = PX \quad \Leftrightarrow \quad x = KR[I_3| - X_O]X$$

Where x are the observed image points (2-D), X is the given control coordinates in the world space (3-D), K is the intrinsic (calibration) 3×3 matrix, R is the 3×3 rotation matrix, $[I_3| - X_O]$ represents the 3×4 extrinsic transformation (combining rotation and translation), and finally, X_O is the 3×1 translation matrix that represents the camera's position relative to the object. However, due to the scale of this project, we are only computing the P matrix. Therefore, we assume that the intrinsic and extrinsic data are combined into the P matrix and are not contained in separate matrices.

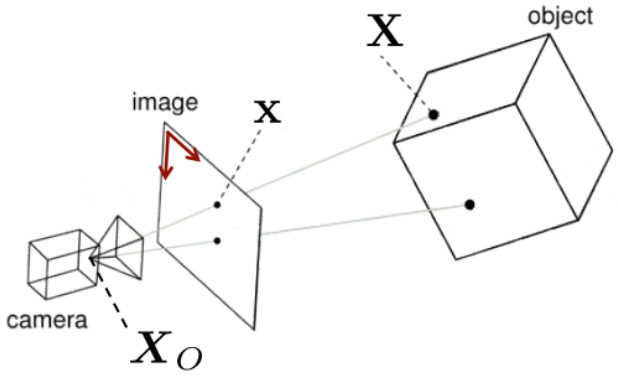

Figure 2: A simple diagram of the camera matrix: x = PX.

To compute the said camera matrix P, we will compute a linear solution:

1. Compute an initial estimate of P by a linear approach:
   (a) Use a similarity transformation matrix T and U to normalize both image points and world space.
   (b) Use direct linear transformation (DLT) to form a $2n \times 12$ matrix A by stacking each corresponding normalized image and world points. The vector p will then contain the entries of the matrix $\tilde{P}$. A solution of $Ap = 0$, with $||p|| = 1$, is obtained from the singular vector with the smallest singular value.
2. Denormalize the normalized camera matrix $\tilde{P}$.

Note: Point correspondences mean $\{X_i \leftrightarrow x_i\}$.

An important observation is that the current DLT (Direct Linear Transformation) approach assumes an affine camera model. This implies that the projection matrix accounts only for linear transformations such as scaling, rotation, and translation without incorporating non-linear effects like perspective. Additionally, DLT is designed for uncalibrated

cameras, meaning that the intrinsic and extrinsic parameters of the camera are unknown. As a result, this assumption can lead to a loss of accuracy in estimating the camera parameters.

In the case of an uncalibrated camera, the projection matrix P contains 11 unknown parameters: five from the intrinsic camera matrix K and six from the extrinsic parameters (the rotation matrix R and the translation vector t). These unknowns need to be solved simultaneously using the point correspondence between 3-D world points and their 2-D image projections.

Using $n \geq 6$ corresponding world and image points, we can simultaneously solve the unknowns and mitigate geometric errors. Each point correspondence contributes two equations (for the x and y image coordinates), so with 6 points, we can acquire 12 equations, which is enough to solve for the 11 unknowns in P. However, we should also note that if all points X_i lie on the same plane, there will be no solution, as the Z_i in the computation shown below will become zero, leading to a rank deficiency.

Lastly, it is crucial to append an extra coordinate of ones to the end of each corresponding point in both the image and world coordinates. This converts the Euclidean coordinates into homogeneous coordinates, allowing us to express points in a projective space. The additional coordinate enables us to represent both 2-D and 3-D points in a manner that can incorporate affine transformations (such as translation, scaling, and rotation) as well as projective transformations (such as perspective projection). This is particularly useful in applications like Homography.

3 Normalization of the image and world points

Normalization is a crucial preprocessing step in estimating the camera matrix, as it helps improve numerical stability. The first step to computing the transformation camera matrix P is to normalize the image points with their respective similarity transformation matrix T. We aim to transform the points x_i into a new set of points $\hat{x}_i$ such that the centroid of $\hat{x}_i$ is shifted to the origin $(0,0)^\top$, and then scaled so that their average distance from the origin becomes $\sqrt{2}$. Refer to Figure 3 below for more clarification.

Given a set of 2-D image points (x, y):

$$x = \{(x_1, y_1), ..., (x_n, y_n)\}$$

The centroid of the points is calculated as:

$$(\bar{x}, \bar{y}) = (\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=1}^{n} y_i), \text{ in other words: taking the mean of the points.}$$

Next, we compute the new set of points $\hat{x}_i$ such that the centroid of the points $\hat{x}_i$ is the coordinate origin $(0,0)^T$:

$$\hat{x}_i = x_i - \bar{x}, \quad \hat{y}_i = y_i - \bar{y}$$

The points are scaled to make the average distance from the origin equal to $\sqrt{2}$. Let the distance of a point from the origin be:

$$d_i = \sqrt{\hat{x}_i^2 + \hat{y}_i^2}$$

The scaling factor s is then computed as:

$$s = \frac{\sqrt{2}}{\frac{1}{n}\sum_{i=1}^{n} d_i}$$

Next, we create the image point transformation matrix T:

$$T = \begin{bmatrix} \mathrm{s} & 0 & -\mathrm{s} \cdot \bar{x} \\ 0 & \mathrm{s} & -\mathrm{s} \cdot \bar{y} \\ 0 & 0 & 1 \end{bmatrix}$$

Finally, we normalize the 2-D image points with the similarity transformation T matrix:

$$\tilde{x_i} = (T \cdot x^T)^T$$

We will need to repeat the process for the corresponding set of 3-D world-space points (x, y, z) such that the average distance from the origin is now $\sqrt{3}$ and a second similarity transformation U matrix to normalized the space point is:

$$U = \begin{bmatrix} \mathrm{s} & 0 & 0 & -\mathrm{s} \cdot \bar{x} \\ 0 & \mathrm{s} & 0 & -\mathrm{s} \cdot \bar{y} \\ 0 & 0 & \mathrm{s} & -\mathrm{s} \cdot \bar{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{X}_i = (U \cdot X^T)^T$$

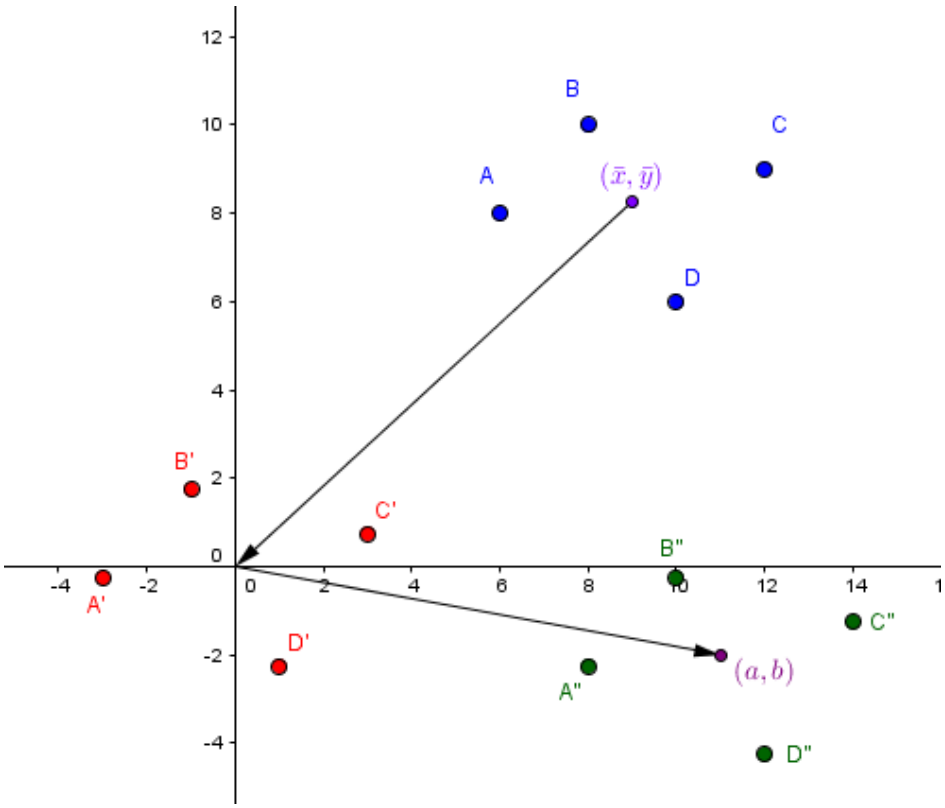

Figure 3: The blue dots are the given set of 2-D image points with their respective centroid $(\bar{x}, \bar{y})$. The red dots are the image points with their centroid shifted to the origin. The green dots are just additional translations that we will not need.

4 Computing the direct linear transformation (DLT)

The second step in computing the P camera matrix involves using the Direct Linear Transformation (DLT) algorithm to calculate the normalized camera projection matrix $\tilde{P}$. We will set the equation $x = PX$ as follows:

$$\tilde{x_i} = \tilde{P_{3\times4}} \tilde{X_i} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \tilde{X_i}$$

Let the following vectors A, B, and C be the transpose of each row vector of P:

$$A = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{bmatrix}, \quad B = \begin{bmatrix} p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \end{bmatrix}, \quad C = \begin{bmatrix} p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}$$

So that we can rewrite the equation as:

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \tilde{x_i} = P\tilde{X}_i = \begin{bmatrix} A^T \\ B^T \\ C^T \end{bmatrix} \tilde{X}_i = \begin{bmatrix} A^T\tilde{X}_i \\ B^T\tilde{X}_i \\ C^T\tilde{X}_i \end{bmatrix}$$

Equivalently:

$$\tilde{x_i} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \begin{bmatrix} A^T\tilde{X}_i \\ B^T\tilde{X}_i \\ C^T\tilde{X}_i \end{bmatrix}$$

Note: x_i and y_i are the 2-D x and y-axis points.

Using the pinhole camera model, the 3-D points $\tilde{P} = (A^T\tilde{X}_i,\ B^T\tilde{X}_i,\ C^T\tilde{X}_i)^T$ can be used to determine its corresponding 2-D image points $\tilde{P}_c = (u_i, v_i)^T$. This relationship is derived through the principles of similar triangles, expressed mathematically as:

$$\frac{u_i}{A^T\tilde{X}_i} = \frac{v_i}{B^T\tilde{X}_i} = \frac{w_i}{C^T\tilde{X}_i}$$

With some simple algebra, we can convert the 3-D point into its corresponding 2-D point:

$$x_i = \frac{u_i}{w_i} = \frac{A^T\tilde{X}_i}{C^T\tilde{X}_i} \Longrightarrow x_iC^T\tilde{X}_i - A^T\tilde{X}_i = 0$$

$$y_i = \frac{v_i}{w_i} = \frac{B^T\tilde{X}_i}{C^T\tilde{X}_i} \Longrightarrow y_iC^T\tilde{X}_i - B^T\tilde{X}_i = 0$$

We can then express x_i and y_i as a system of linear equations:

$$\begin{cases} -X_i^TA + x_iX_i^TC = 0 \\ -X_i^TB + y_iX_i^TC = 0 \end{cases}$$

Now, we can collect the elements of $\tilde{P}$ within a parameter vector p:

$$p = p_k = \begin{bmatrix} A \\ B \\ C \end{bmatrix} = vec(\tilde{P}^T)$$

Observe that $[A\ B\ C]^T$ is composed of the rows of $\tilde{P}$, arranged as column vectors stacked vertically to form a 12×1 vector. Consequently, the system of linear equations can be reformulated in terms of the vector p:

$$a_{x_i}^Tp = 0$$

$$a_{y_i}^Tp = 0$$

where:

$$a_{x_i}^T = (-\tilde{X_i}^T, 0^T, x_i\tilde{X_i}^T) = (-X_i, -Y_i, -Z_i, -1, 0, 0, 0, 0, x_iX_i, x_iY_i, x_iZ_i, x_i)$$

$$a_{y_i}^T = (0^T, -\tilde{X_i}^T, xy_i\tilde{X_i}^T) = (0, 0, 0, 0, -X_i, -Y_i, -Z_i, -1, y_iX_i, y_iY_i, y_iZ_i, y_i)$$

Now, stacking everything together to formulate $Ap = 0$:

$$\begin{bmatrix} a_{x_1}^T \\ a_{y_2}^T \\ \vdots \\ a_{x_i}^T \\ a_{y_i}^T \\ \vdots \\ a_{x_n}^T \\ a_{y_n}^T \end{bmatrix} p = Ap = 0$$

Note: A is a $2n \times 12$ matrix, and p is a 12×1 column vector.

Solving $Ap = 0$ is equivalent to finding the null space of A. This can be achieved using singular value decomposition (SVD), where A is decomposed as $A = P\Sigma Q^T$ where $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$ contains the singular values such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n > 0$. By selecting the last column of the orthogonal matrix Q, denoted as $p = q_{12}$, we obtain the singular vector corresponding to the smallest singular value, which minimizes geometric error. We also need to reshape the p vector back into the 3×4 matrix $\tilde{P}$:

$$p = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ \vdots \\ p_{34} \end{bmatrix} \implies \tilde{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}$$

5 De-normalization to acquire P

After performing normalization and DLT, the final step in obtaining the camera matrix P is straightforward. We need to recover the original (unnormalized) coordinates from $\tilde{P}$ by:

$$P = T^{-1}\tilde{P}U$$

6 Conclusion

The camera matrix P has many applications, particularly in game development. One example is mapping the 3-D world onto a player's camera view or using it to create aim assistance for AI enemies. Additionally, obtaining the camera matrix P is the first step in computing other key concepts in computer vision, such as Homography and the fundamental matrix. While this project focused on computing the elements of the camera matrix, there are additional steps beyond its scope, such as minimizing geometric error using algorithms like Levenberg–Marquardt.

7 MATLAB Code

```
1  % @brief: Computes the similarity transformation matrix T with its
2  % respective normalized image points and the similarity transformation
3  % matrix U with its respective normalized world points.
4  %
5  % @param x: 2D image points, a matrix of size (n x 3) where each row is
6  % [x, y, 1]
7  % @param X: 3D world points, a matrix of size (n x 4) where each row is
8  % [X, Y, Z, 1]
9  % return [T, x_norm, U, X_norm]: The two similarity trasnformation matrices
10 % and the normalized image and world points.
11 function [T, x_norm, U, X_norm] = normalizePoints(x, X)
12     % Ensure there are n >= 6 points to compute camera matrix P
13     X_n = size(X, 1);
14     x_n = size(x, 1);
15     if X_n < 6 || x_n < 6
16         error("At least 6 points (both X & x) are required to compute the camera matrix.");
17     end
18
19     % Ensure 3D world points matrix has 4 columns &
20     % 2D image points matrix has 3 columns
21     if size(X, 2) ~= 4 || size(x, 2) ~= 3
22         error("Incorrect dimensions. X should have 4 columns & x should have 3 columns.");
23     end
24
25     % Acquire the center point of all mapped x (2D) points (centroid)
26     centroid_x = mean(x(:, 1:2));
27
28     % Set centroid_x's coordinate origin to (0, 0)^T
29     x_centered = x(:, 1:2) - centroid_x;
30
31     % Compute x_centered's average distance from origin
32     avg_dist_x = mean(sqrt(sum(x_centered.^2, 2)));
33
```

```
34      % Scale the points to make the average distance from origin
            to sqrt(2)
35      scale_x = sqrt(2) / avg_dist_x;
36
37      % Create the image points transformation matrix T
38      T = [scale_x, 0, -scale_x * centroid_x(1);
39           0, scale_x, -scale_x * centroid_x(2);
40           0, 0, 1];
41
42      % Normalize 2D points with T
43      x_norm = (T * x')';
44
45      % Acquire the center point of all mapped X (3D) points (
            centroid)
46      centroid_X = mean(X(:, 1:3));
47
48      % Set centroid_X's coordinate origin to (0, 0)^T
49      X_centered = X(:, 1:3) - centroid_X;
50
51      % Compute X_centered's average distance from origin
52      avg_dist_X = mean(sqrt(sum(X_centered.^2, 2)));
53
54      % Scale the points to make the average distance from origin
            to sqrt(2)
55      scale_X = sqrt(3) / avg_dist_X;
56
57      % Create the space points transformation matrix U
58      U = [scale_X, 0, 0, -scale_X * centroid_X(1);
59           0, scale_X, 0, -scale_X * centroid_X(2);
60           0, 0, scale_X, -scale_X * centroid_X(3);
61           0, 0, 0, 1];
62
63      % Normalize 3D points with U
64      X_norm = (U * X')';
65  end
66
67  % @brief: Computes the camera matrix P given a set of 6 or more
        3D points
68  % with its respective 2D point on an image plane for an
        uncalibrated
69  % camera. Assuming an affine camera model non-respective to
        perspective.
70  %
71  % @param x: 2D image points, a matrix of size (n x 3) where
        each row is
```

```
% [x, y, 1]
% @param X: 3D world points, a matrix of size (n x 4) where each row is
% [X, Y, Z, 1]
% return P: The computed camera projection matrix (3 x 4)
function P = computeCameraMatrix(x, X)
    % Ensure there are n >= 6 points to compute camera matrix P
    X_n = size(X, 1);
    x_n = size(x, 1);
    if X_n < 6 || x_n < 6
        error("At least 6 points (both X & x) are required to compute the camera matrix.");
    end

    % Ensure 3D world points matrix has 4 columns &
    % 2D image points matrix has 3 columns
    if size(X, 2) ~= 4 || size(x, 2) ~= 3
        error("Incorrect dimensions. X should have 4 columns & x should have 3 columns.");
    end

    n = X_n;

    % Construct the matrix A
    A = zeros(2 * n, 12); % Each point has corresponding (a_xi)^T & (a_yi)^T

    for i = 1:n
        X_i = X(i, :); % Current 3D point [X, Y, Z, 1] (X_i)
        u_i = x(i, 1); % Image point x-coordinate (x_i)
        v_i = x(i, 2); % Image point y-coordinate (y_i)

        % Compute the corresponding (a_xi)^T row:
        % (a_xi)^T = [-X_i, -Y_i, -Z_i, -1, 0, 0, 0, 0, (y_i * X_i), (y_i *
        % Y_i), (y_i * Z_i), y_i]
        A(2 * i - 1, :) = [-X_i, zeros(1, 4), u_i * X_i];

        % Compute the corresponding (a_yi)^T row:
        % (a_xi)^T = [0, 0, 0, 0, -X_i, -Y_i, -Z_i, -1, (x_i * X_i), (x_i *
        % Y_i), (x_i * Z_i), x_i]
        A(2 * i, :) = [zeros(1, 4), -X_i, v_i * X_i];
    end

```

```
    % Solving for Ap = 0 using Singular Value Decomposition (SVD)
    [~, ~, Q] = svd(A);

    % Take the last column of V as solution to minimize    = _ii
    p = Q(:, end);

    % Reshape p into the 3x4 camera matrix P
    P = reshape(p, 4, 3)';
end

% Define 3D world points (m x 4), homogeneous coordinates
X = [0, 0, 0, 1;
     1, 0, 0, 1;
     1, 1, 0, 1;
     0, 1, 0, 1;
     0, 0, 1, 1;
     1, 0, 1, 1];

% Define corresponding 2D image points (m x 3), homogeneous coordinates
x = [100, 200, 1;
     200, 200, 1;
     200, 300, 1;
     100, 300, 1;
     120, 220, 1;
     220, 220, 1];

[T, x_normalized, U, X_normalized] = normalizePoints(x, X);

P_norm = computeCameraMatrix(x_normalized, X_normalized);
disp("Normalized Camera Projection Matrix P:")
disp(P_norm);

% Acquire original camera matrix by denormalize the normalized camera matrix
P = inv(T) * P_norm * U;
disp("Final camera matrix P:");
disp(P);

% Verification: Project the 3D world point's 2D image point using the final camera matrix
x_projected = (P * X')';
for i = 1:size(x_projected, 1)
```

```
151     x_projected(i, :) = x_projected(i, :) / x_projected(i, 3);
          % Normalize
152 end
153 disp('Projected 2D points (x_projected):');
154 disp(x_projected);
155
156 % Computing the transfer geometric error. Side note: textbook's
      d(x, y) is the
157 % Euclidean distance between the inhomogeneous points x and y.
158 transferError = 0;
159 for i = 1:size(X, 1)
160    difference = x(i, 1:2) - x_projected(i, 1:2);
161    transferError = transferError + norm(difference)^2;
162 end
163 disp("Transfer Error:" );
164 disp(transferError);
```

References

[1] "Finding the center of an arbitrary set of points in two dimensions." Math Stack Exchange. [Online]. Available: https://math.stackexchange.com/questions/1801867/finding-the-centre-of-an-abritary-set-of-points-in-two-dimensions. [Accessed: 22-Dec-2024].

[2] Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in Computer Vision*. Cambridge University Press.

[3] Kitani, K. (2017). "Camera Matrix," Carnegie Mellon University. [Online]. Available: https://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf. [Accessed: 22-Dec-2024].

[4] Stachniss, Cyril. "Camera Calibration: Direct Linear Transform," University of Bonn, 2021. [Online]. Available: https://www.ipb.uni-bonn.de/html/teaching/photo12-2021/2021-pho1-21-DLT.pptx.pdf. [Accessed: 22-Dec-2024].