CSC 422/522 Computer Vision and Pattern Recognition

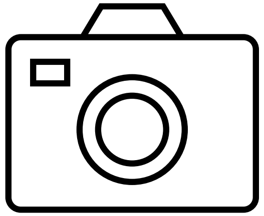
Camera Calibration

2026 Spring

South Dakota State University

Today

- Recap: Camera Model
  - Intrinsic Matrix *K*
  - Extrinsic Matrix *T*
  - Putting Everything Together (Projection Matrix *P*)
- Practical Setup: Google Colab
  - Brief Overview of Google Colab
  - Demo of Camera Calibration in Google Colab
- Theory Behind Camera Calibration
  - How to recover *P, K, R, t*
  - How to handle camera distortion



OpenCV

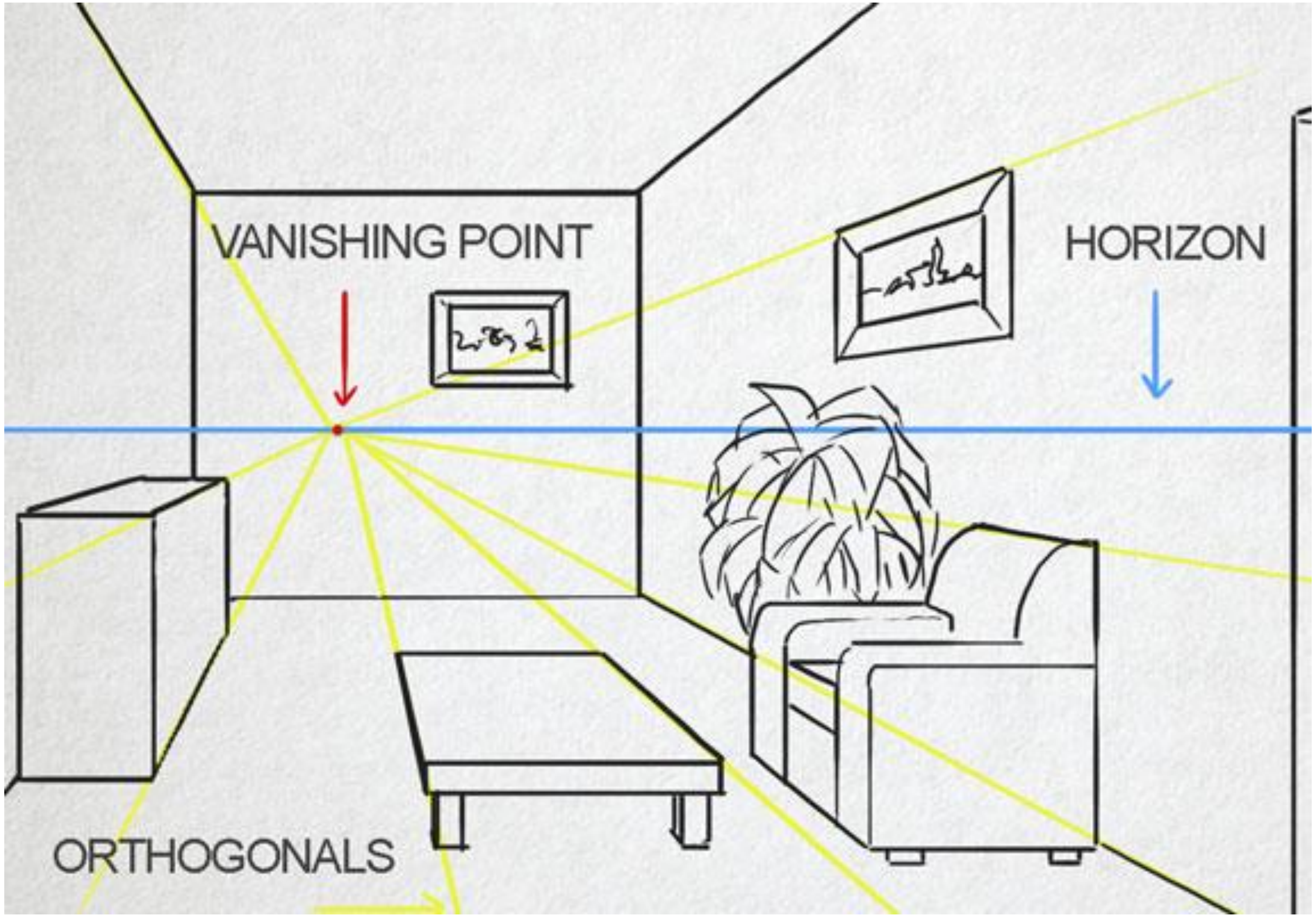


Image source: https://thevirtualinstructor.com/onepointperspective.html

Recap: Camera Model

Camera and World Coordinate Systems

Camera Coordinate System

- It is attached to the camera, and by convention:
- Z-axis along the viewing direction
- X, Y axes aligned with the image plane

World Coordinate System

- It is a reference frame you choose to describe the scene, which has
- An origin (chosen arbitrarily)
- Axes (Chosen arbitrarily)

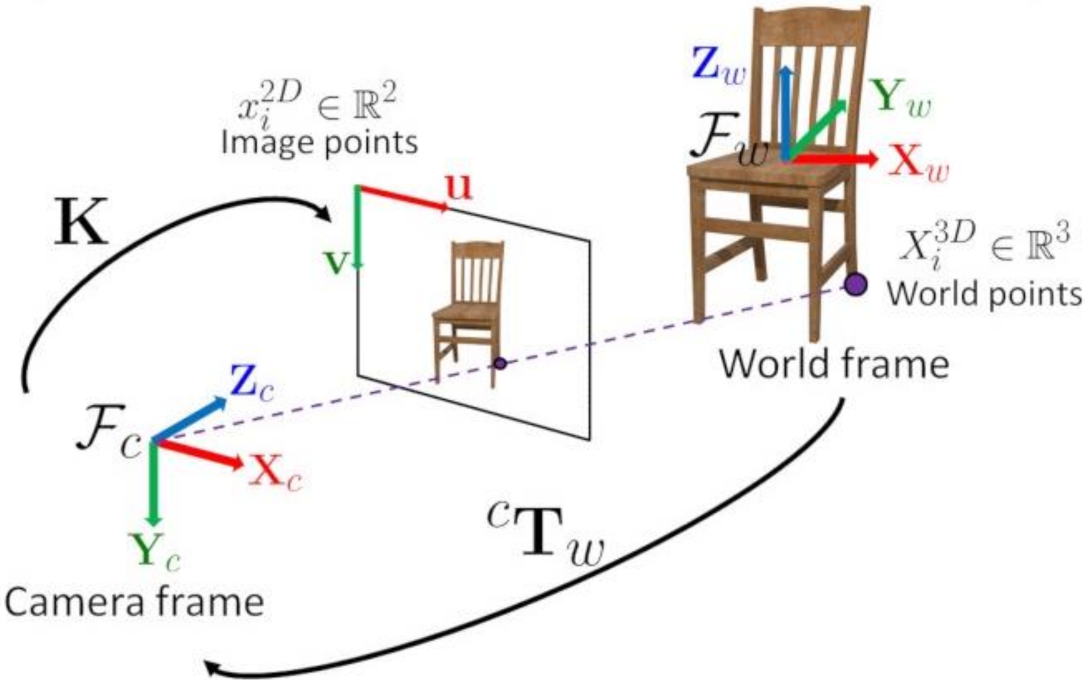


Image source: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

World-to-Camera Transformation: The Extrinsic Matrix *T*

$$T_{n\times n} = \begin{bmatrix} R_{n\times n} & t_{n\times 1} \\ 0_{1\times n} & 1 \end{bmatrix}$$

Rewriting using homogenous coordinates:

$$\boxed{\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}}$$

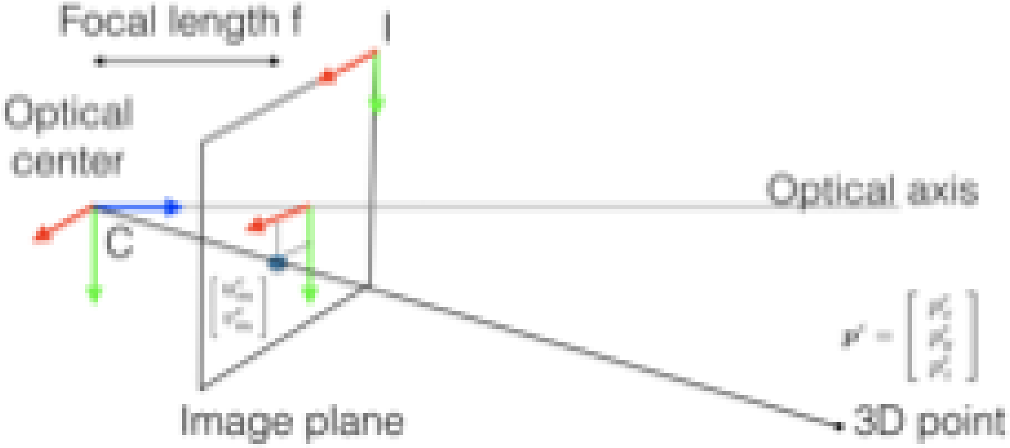
Extrinsic Matrix: $M_{ext} = \begin{bmatrix} R_{3\times 3} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

The Full Equation from 3D Point in Camera Coordinate System to Pixel

$$u_m^c = f\frac{p_x^c}{p_z^c} \qquad v_m^c = f\frac{p_y^c}{p_z^c} \tag{11.1}$$

$$u^I = s_x u_m^c + o_x \qquad v^I = s_y v_m^c + o_y \tag{11.5}$$



And denote: $f_x = s_x f$, $f_y = s_y f$

We can get: $u^I = s_x f\frac{p_x^c}{p_z^c} + o_x = f_x\frac{p_x^c}{p_z^c} + o_x$

$$v^I = s_y f\frac{p_y^c}{p_z^c} + o_y = f_y\frac{p_y^c}{p_z^c} + o_y$$

The Intrinsic Matrix *K*

$$p_z^c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \end{bmatrix}$$

Note: in this course, we assume the pixels are not skewed. Otherwise, the value in the first row, second column will be non-zero.

Notations we use:

- $[o_x, o_y]$ is called the principal point
- f_x is the focal length in horizontal pixels
- f_y is the focal length in vertical pixels

Combining Extrinsic and Intrinsic Matrices

Image Coordinates

$$X_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Intrinsic

Camera Coordinates

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

Extrinsic

World Coordinates

$$X_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

Using homogenous coordinates:

$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix}$ represent the same point

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

Putting It All Together: Projection Matrix *P*

Camera to Pixel

$$\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \tilde{w}\end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0\\ 0 & f_y & o_y & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_c\\ y_c\\ z_c\\ 1\end{bmatrix}$$

$$\tilde{\mathbf{u}} = M_{int}\,\tilde{\mathbf{x}}_{\mathbf{c}}$$

World to Camera

$$\begin{bmatrix}x_c\\ y_c\\ z_c\\ 1\end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x\\ r_{21} & r_{22} & r_{23} & t_y\\ r_{31} & r_{32} & r_{33} & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_w\\ y_w\\ z_w\\ 1\end{bmatrix}$$

$$\tilde{\mathbf{x}}_{\mathbf{c}} = M_{ext}\tilde{\mathbf{x}}_{\mathbf{w}}$$

Combining the above two equations, we get the full projection matrix P:

$$\boxed{\tilde{\mathbf{u}} = M_{int}M_{ext}\,\tilde{\mathbf{x}}_{\mathbf{w}} = P\,\tilde{\mathbf{x}}_{\mathbf{w}}}$$

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

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

Why study the projection matrix

- Computer Graphics: use ***P*** render 3D scenes onto 2D images

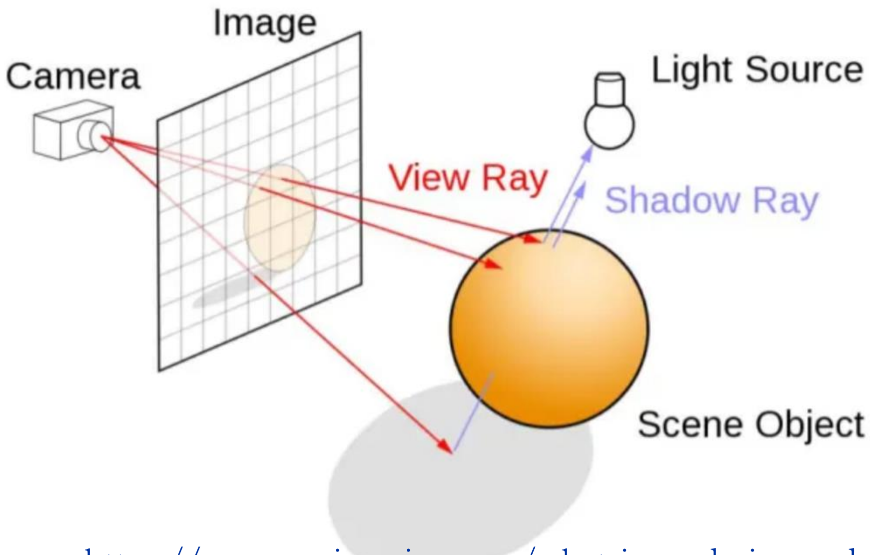


Image source: https://www.engineering.com/what-is-rendering-and-why-is-it-important-for-engineers/

- Computer Vision: Estimate ***P*** from 2D images

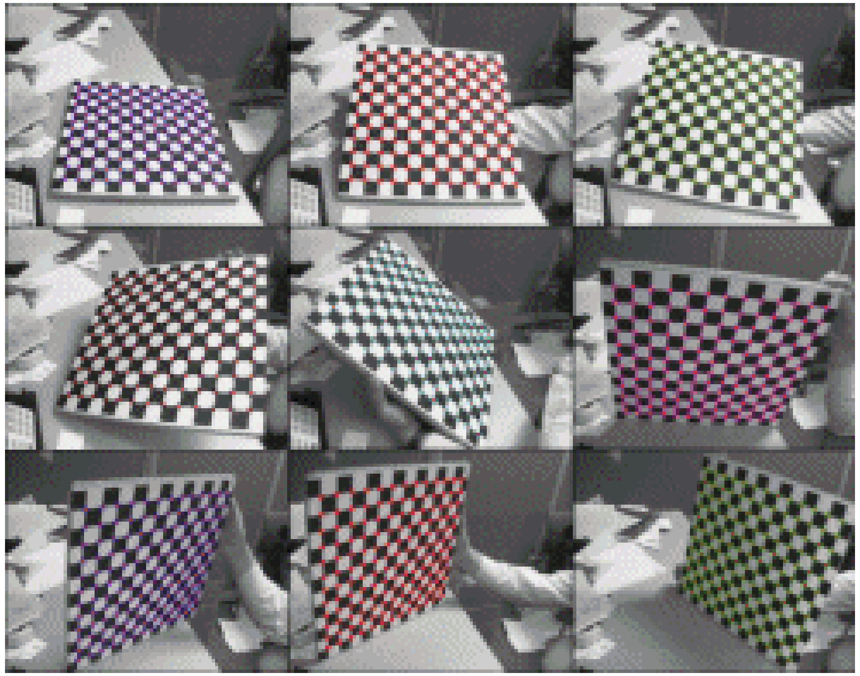
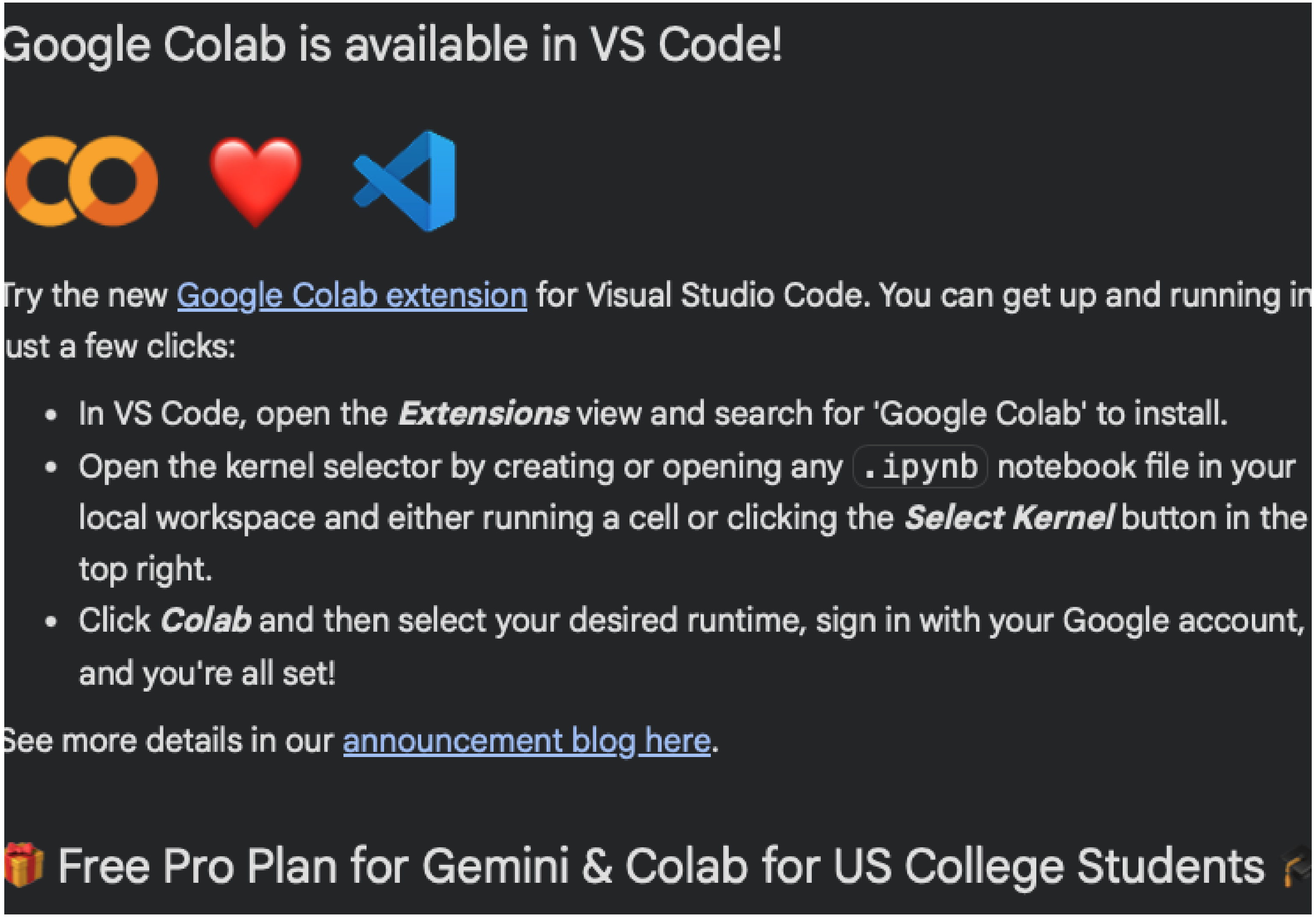


Image source: https://robots.stanford.edu/cs223b04/JeanYvesCalib/

SOUTH DAKOTA STATE UNIVERSITY



Google Colab is available in VS Code!

Try the new Google Colab extension for Visual Studio Code. You can get up and running in ust a few clicks:

- In VS Code, open the ***Extensions*** view and search for 'Google Colab' to install.
- Open the kernel selector by creating or opening any `.ipynb` notebook file in your local workspace and either running a cell or clicking the ***Select Kernel*** button in the top right.
- Click ***Colab*** and then select your desired runtime, sign in with your Google account, and you're all set!

See more details in our announcement blog here.

🎁 Free Pro Plan for Gemini & Colab for US College Students

Screenshot from https://colab.research.google.com/#

Google Colab: Quick Start!

What is Colab?

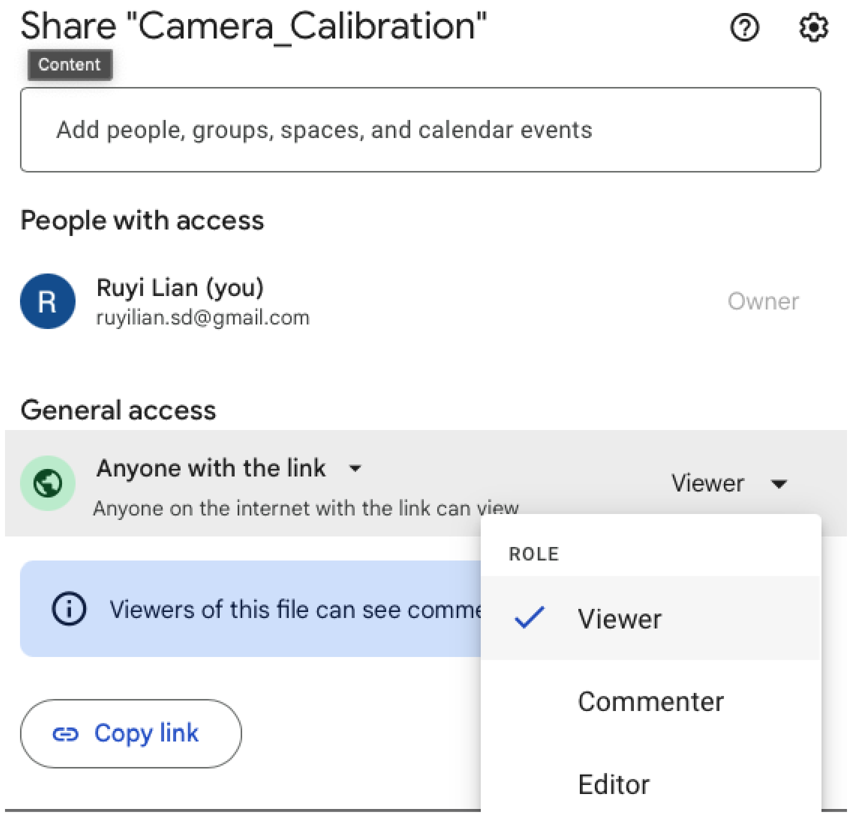
- Colab, or "Colaboratory", allows you to write and execute Python in your browser, with
  - Zero configuration required
  - Access to GPUs free of charge
  - Easy sharing
- If you're familiar with the popular Jupyter project, you can think of Colab as a Jupyter notebook stored in Google Drive.
- A notebook is a list of cells. Cells contain either explanatory text or executable code and its output.

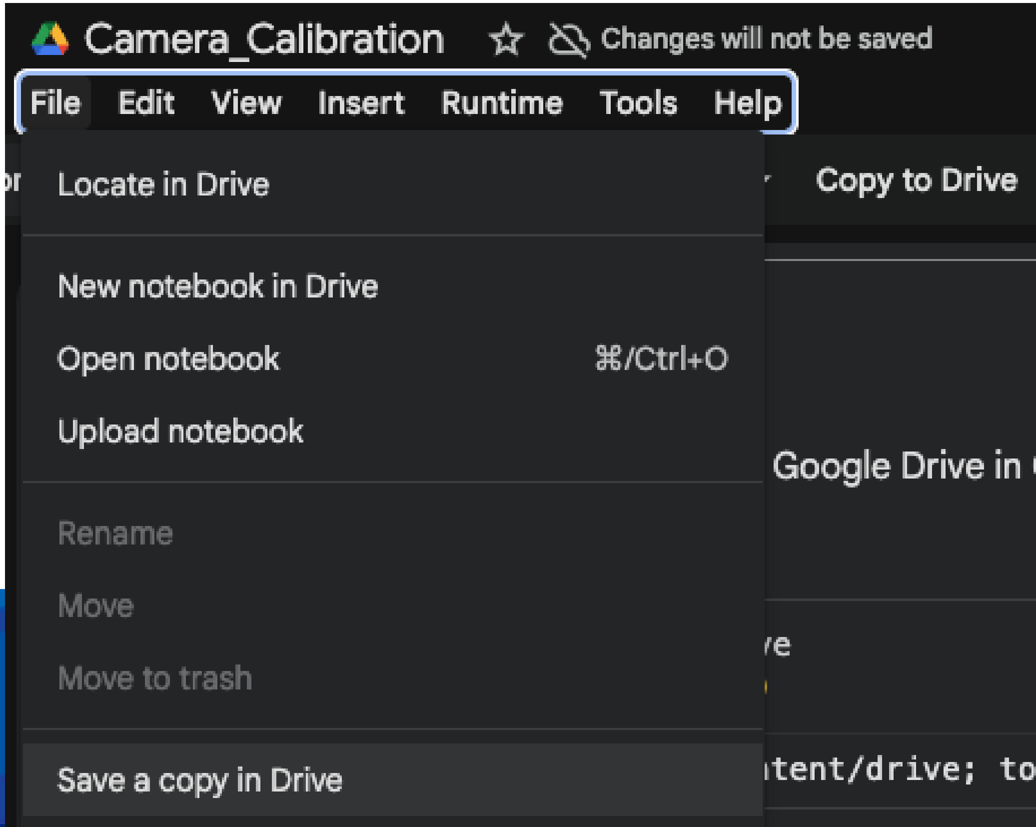
In-Class Practice 1: Run the shared Google Colab Notebook

- In this activity, we will use the official Google Colab tutorial to gain hands-on experience, including:
  - Cells: code cells, text cells, adding and moving cells
  - Run Python codes
- Links:
  - https://colab.research.google.com/notebooks/basic_features_overview.ipynb
  - https://colab.research.google.com/#

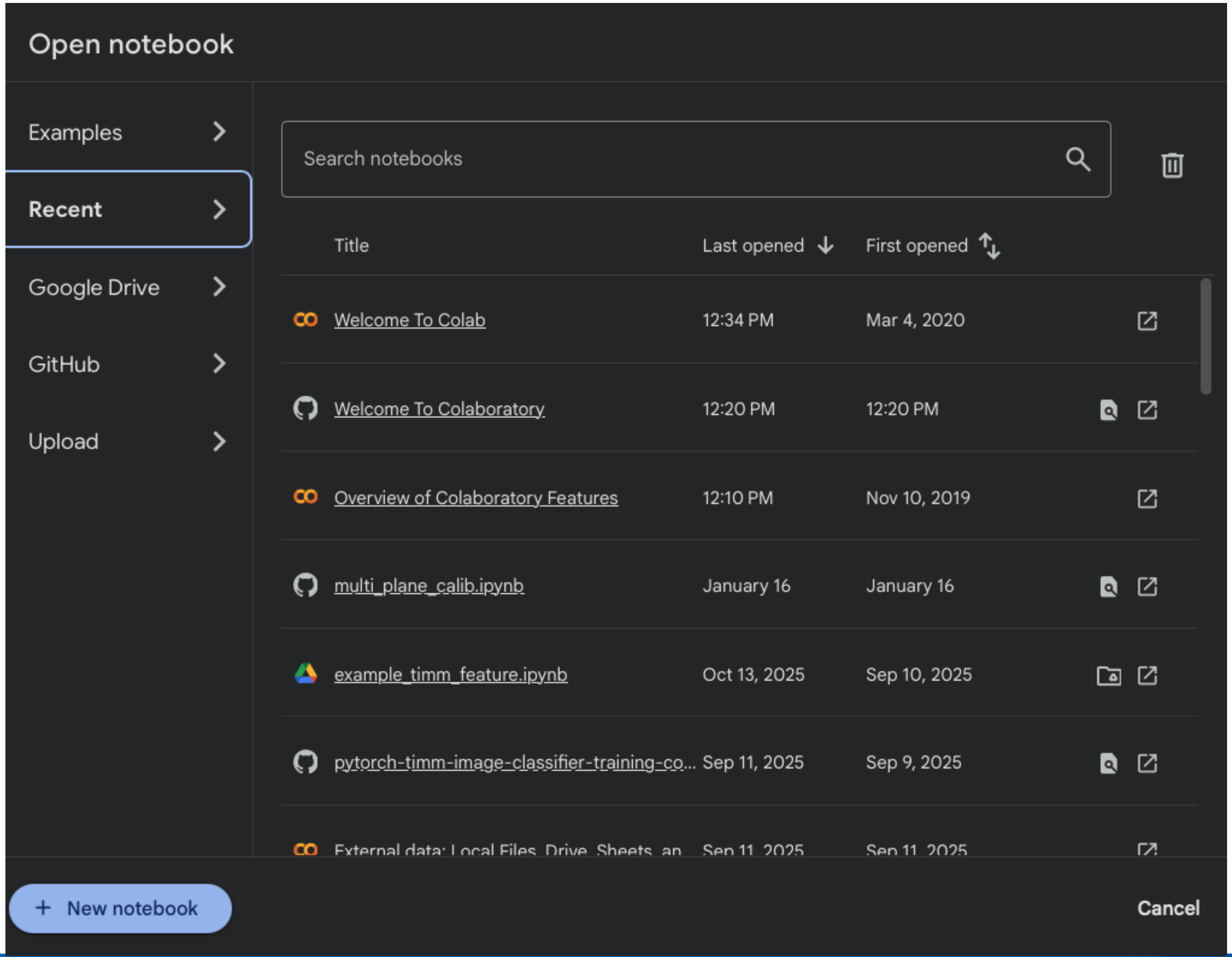
What happens when you share a Colab notebook?

- Usually, we share a view-only link.
- Others can open and run all cells.
- Others can modify the notebook, but: all changes are temporary and will be lost if they close the tab unless they save a copy.

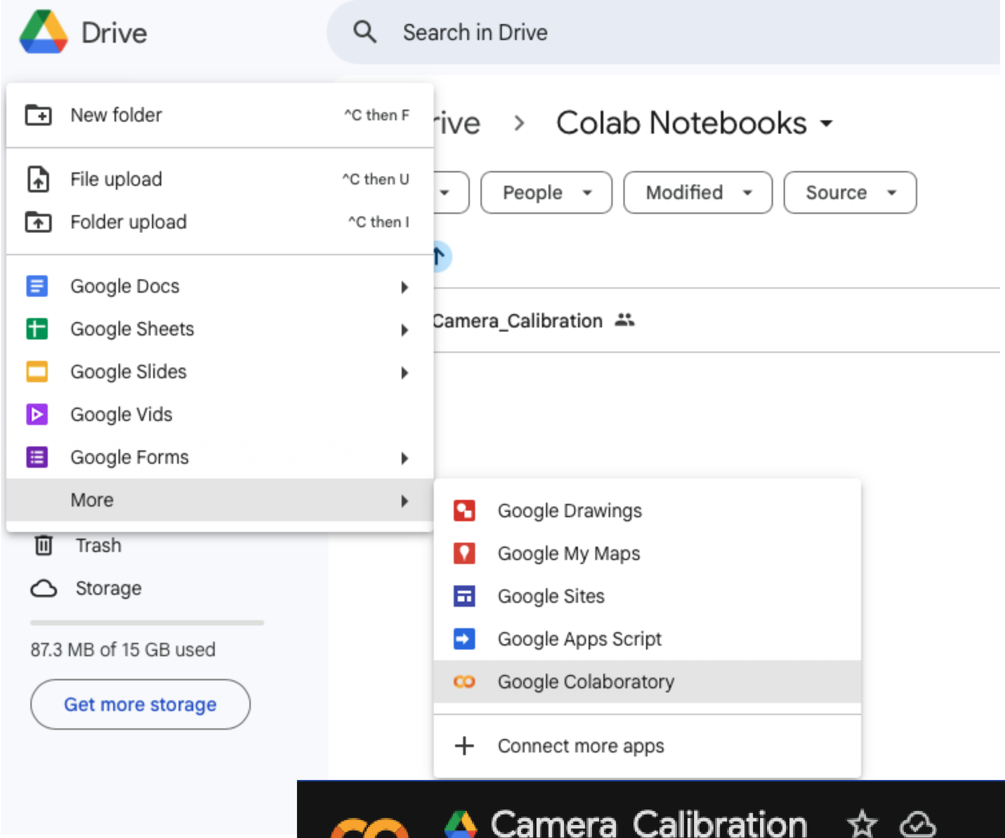


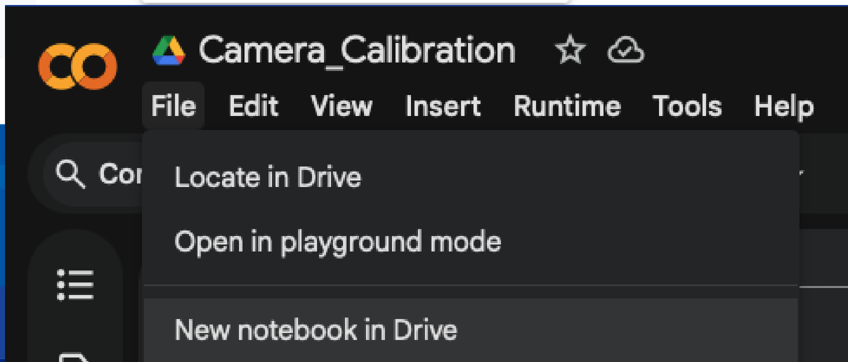


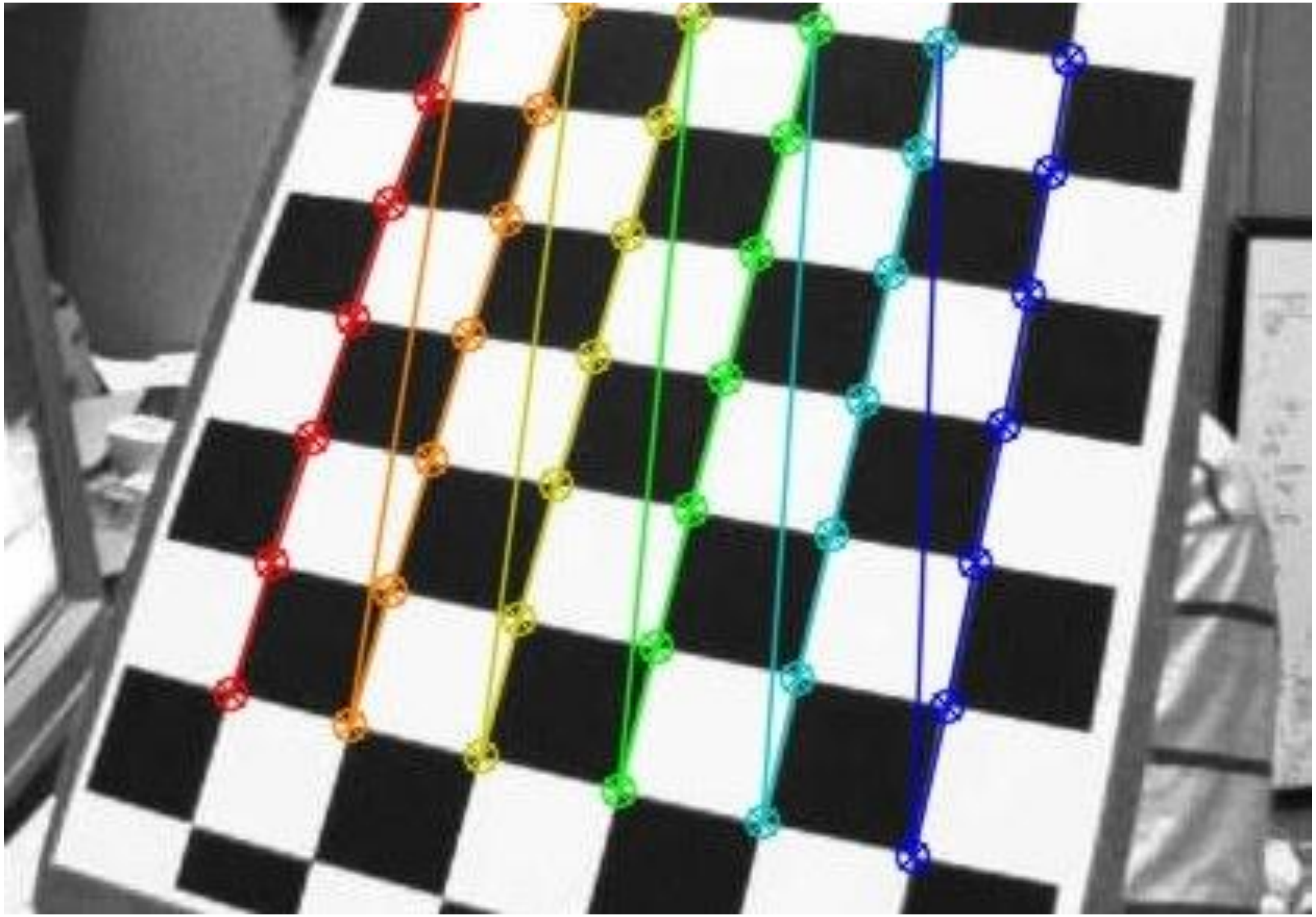
In-Class Practice 2: Create a New Colab Notebook



There are multiple ways…







Camera Calibration: a OpenCV Demo

Reference: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

Colab Notebook Demo

- In this demo, we will learn how to:
  - Load our customized data from Google Drive
  - Use OpenCV to find all corners in calibration plane
  - How to get the camera parameters
  - Applications including getting undistorted images and projecting a cube
- Link: https://colab.research.google.com/drive/119FBVHOopIQkCS3LDUNcUMI8GVycSTCb?usp=sharing

Assignment 1: Camera Calibration (Warm-Up Project)

- Goal: Get familiar with data collection, Google Colab, and Python through a simple camera calibration task.
- Print a calibration pattern (chessboard).
- Take at least 12 images and store in your google drive
- Adapt the demo code to compute camera parameters
- Keep the outputs of all your code cells, and download your colab notebook as a ".ipynb" file (File > Download > Download.ipynb)
- Use text cells to briefly describe the difficulties you have met in this warm-up project (e.g., python syntax, image quality, etc.)

How to get Project Matrix *P*

Camera to Pixel

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{u}} = M_{int}\, \tilde{\mathbf{x}}_c$$

World to Camera

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{x}}_c = M_{ext} \tilde{\mathbf{x}}_w$$

Combining the above two equations, we get the full projection matrix P:

$$\tilde{\mathbf{u}} = M_{int} M_{ext}\, \tilde{\mathbf{x}}_w = P\, \tilde{\mathbf{x}}_w$$

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

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

Credits: the slides of this section are adapted from CSC 492/592 (Spring 2024)

Camera Calibration: the Mathematical Model

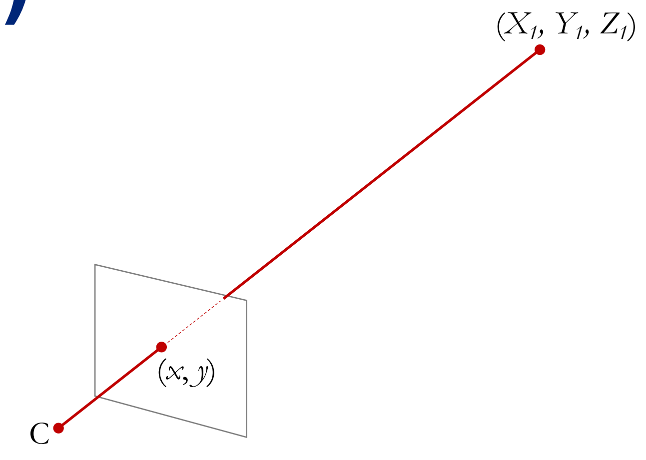
- What do we need?
  - 3D world points and their corresponding 2D image points
- How do we do?
  - Get the correspondences
  - Decompose *P*

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- After you calibrate a camera, then …
  - evaluate the accuracy of the estimated parameters:
    - Plot the relative locations of the camera and the calibration pattern
    - Calculate the reprojection errors
    - Calculate the parameter estimation errors

Getting Linear Equations (part 1)

$$P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

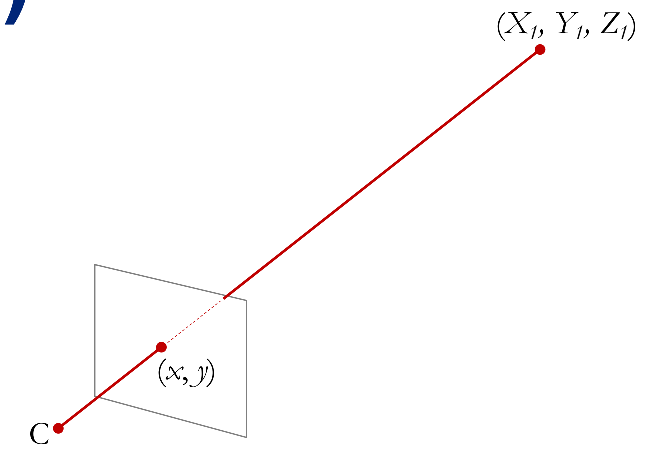


$u = p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3$
$v = p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7$
$w = p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}$

$x = u/w = (p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3) \;/\; (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$
$y = v/w = (p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7) \;/\; (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$

Getting Linear Equations (part 2)

$$P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



$$u = p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3$$
$$v = p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7$$
$$w = p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}$$

$$\boxed{x = u/w = (p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3) \;/\; (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})}$$
$$y = v/w = (p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7) \;/\; (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$$

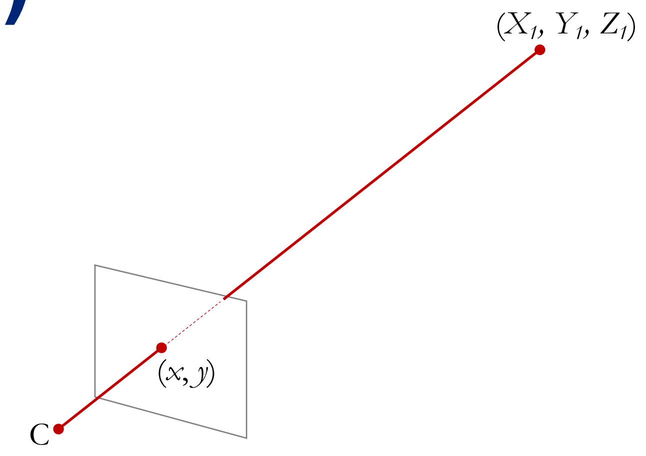
multiply $(p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$ on both sides

$$x \cdot (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}) = (p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3)$$
$$x \cdot (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}) - (p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3) = 0$$

$$\boxed{- p_0 \cdot X - p_1 \cdot Y - p_2 \cdot Z - p_3 + 0 \cdot p_4 + 0 \cdot p_5 + 0 \cdot p_6 + 0 \cdot p_7 + x \cdot p_8 \cdot X + x \cdot p_9 \cdot Y + x \cdot p_{10} \cdot Z + x \cdot p_{11} = 0}$$

Getting Linear Equations (part 3)

$$P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



$$u = p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3$$
$$v = p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7$$
$$w = p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}$$

$$x = u/w = (p_0 \cdot X + p_1 \cdot Y + p_2 \cdot Z + p_3) \ / \ (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$$
$$\boxed{y = v/w = (p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7) \ / \ (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})}$$

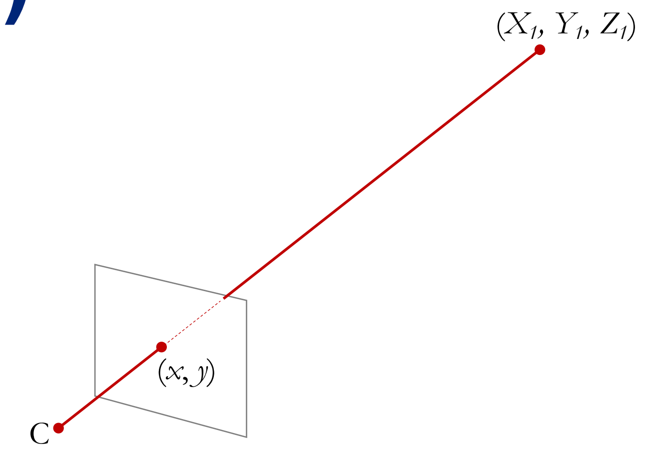
multiply $(p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11})$ on both sides

$$y \cdot (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}) = (p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7),$$
$$y \cdot (p_8 \cdot X + p_9 \cdot Y + p_{10} \cdot Z + p_{11}) - (p_4 \cdot X + p_5 \cdot Y + p_6 \cdot Z + p_7) = 0,$$

$$\boxed{0 \cdot p_0 + 0 \cdot p_1 + 0 \cdot p_2 + 0 \cdot p_3 - p_4 \cdot X - p_5 \cdot Y - p_6 \cdot Z - p_7 + y \cdot p_8 \cdot X + y \cdot p_9 \cdot Y + y \cdot p_{10} \cdot Z + y \cdot p_{11} = 0}$$

Getting Linear Equations (part 4)

$$P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

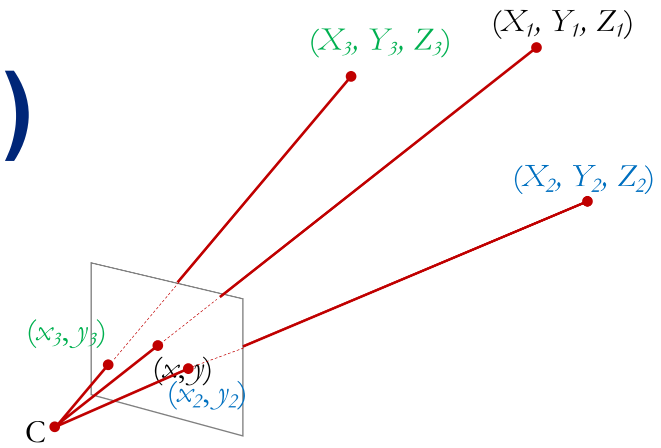


$$- p_0 \cdot X_1 - p_1 \cdot Y_1 - p_2 \cdot Z_1 - p_3 + 0 \cdot p_4 + 0 \cdot p_5 + 0 \cdot p_6 + 0 \cdot p_7 + x \cdot p_8 \cdot X_1 + x \cdot p_9 \cdot Y_1 + x \cdot p_{10} \cdot Z_1 + x \cdot p_{11} = 0$$

$$0 \cdot p_0 + 0 \cdot p_1 + 0 \cdot p_2 + 0 \cdot p_3 - p_4 \cdot X_1 - p_5 \cdot Y_1 - p_6 \cdot Z_1 - p_7 + y \cdot p_8 \cdot X_1 + y \cdot p_9 \cdot Y_1 + y \cdot p_{10} \cdot Z_1 + y \cdot p_{11} = 0$$

We have already obtained two linear equations from one pair of correspondences between 3D world point and 2D image point ☺ what's next?

Getting Linear Equations (part 5)



$$P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$-p_0\cdot X_1 - p_1\cdot Y_1 - p_2\cdot Z_1 - p_3 + 0\cdot p_4 + 0\cdot p_5 + 0\cdot p_6 + 0\cdot p_7 + x\cdot p_8\cdot X_1 + x\cdot p_9\cdot Y_1 + x\cdot p_{10}\cdot Z_1 + x\cdot p_{11} = 0$$

$$0\cdot p_0 + 0\cdot p_1 + 0\cdot p_2 + 0\cdot p_3 - p_4\cdot X_1 - p_5\cdot Y_1 - p_6\cdot Z_1 - p_7 + y\cdot p_8\cdot X_1 + y\cdot p_9\cdot Y_1 + y\cdot p_{10}\cdot Z_1 + y\cdot p_{11} = 0$$

$$-p_0\cdot X_2 - p_1\cdot Y_2 - p_2\cdot Z_2 - p_3 + 0\cdot p_4 + 0\cdot p_5 + 0\cdot p_6 + 0\cdot p_7 + x_2\cdot p_8\cdot X_2 + x_2\cdot p_9\cdot Y_2 + x_2\cdot p_{10}\cdot Z_2 + x_2\cdot p_{11} = 0$$

$$0\cdot p_0 + 0\cdot p_1 + 0\cdot p_2 + 0\cdot p_3 - p_4\cdot X_2 - p_5\cdot Y_2 - p_6\cdot Z_2 - p_7 + y_2\cdot p_8\cdot X_2 + y_2\cdot p_9\cdot Y_2 + y_2\cdot p_{10}\cdot Z_2 + y_2\cdot p_{11} = 0$$

$$-p_0\cdot X_3 - p_1\cdot Y_3 - p_2\cdot Z_3 - p_3 + 0\cdot p_4 + 0\cdot p_5 + 0\cdot p_6 + 0\cdot p_7 + x_3\cdot p_8\cdot X_3 + x_3\cdot p_9\cdot Y_3 + x_3\cdot p_{10}\cdot Z_3 + x_3\cdot p_{11} = 0$$

$$0\cdot p_0 + 0\cdot p_1 + 0\cdot p_2 + 0\cdot p_3 - p_4\cdot X_3 - p_5\cdot Y_3 - p_6\cdot Z_3 - p_7 + y_3\cdot p_8\cdot X_3 + y_3\cdot p_9\cdot Y_3 + y_3\cdot p_{10}\cdot Z_3 + y_3\cdot p_{11} = 0$$

We Have Obtained a System of Linear Equations!

$$-p_0 \cdot X_1 - p_1 \cdot Y_1 - p_2 \cdot Z_1 - p_3 + 0 \cdot p_4 + 0 \cdot p_5 + 0 \cdot p_6 + 0 \cdot p_7 + x \cdot p_8 \cdot X_1 + x \cdot p_9 \cdot Y_1 + x \cdot p_{10} \cdot Z_1 + x \cdot p_{11} = 0$$

$$0 \cdot p_0 + 0 \cdot p_1 + 0 \cdot p_2 + 0 \cdot p_3 - p_4 \cdot X_1 - p_5 \cdot Y_1 - p_6 \cdot Z_1 - p_7 + y \cdot p_8 \cdot X_1 + y \cdot p_9 \cdot Y_1 + y \cdot p_{10} \cdot Z_1 + y \cdot p_{11} = 0$$

$$-p_0 \cdot X_2 - p_1 \cdot Y_2 - p_2 \cdot Z_2 - p_3 + 0 \cdot p_4 + 0 \cdot p_5 + 0 \cdot p_6 + 0 \cdot p_7 + x_2 \cdot p_8 \cdot X_2 + x_2 \cdot p_9 \cdot Y_2 + x_2 \cdot p_{10} \cdot Z_2 + x_2 \cdot p_{11} = 0$$

$$0 \cdot p_0 + 0 \cdot p_1 + 0 \cdot p_2 + 0 \cdot p_3 - p_4 \cdot X_2 - p_5 \cdot Y_2 - p_6 \cdot Z_2 - p_7 + y_2 \cdot p_8 \cdot X_2 + y_2 \cdot p_9 \cdot Y_2 + y_2 \cdot p_{10} \cdot Z_2 + y_2 \cdot p_{11} = 0$$

$$-p_0 \cdot X_3 - p_1 \cdot Y_3 - p_2 \cdot Z_3 - p_3 + 0 \cdot p_4 + 0 \cdot p_5 + 0 \cdot p_6 + 0 \cdot p_7 + x_3 \cdot p_8 \cdot X_3 + x_3 \cdot p_9 \cdot Y_3 + x_3 \cdot p_{10} \cdot Z_3 + x_3 \cdot p_{11} = 0$$

$$0 \cdot p_0 + 0 \cdot p_1 + 0 \cdot p_2 + 0 \cdot p_3 - p_4 \cdot X_3 - p_5 \cdot Y_3 - p_6 \cdot Z_3 - p_7 + y_3 \cdot p_8 \cdot X_3 + y_3 \cdot p_9 \cdot Y_3 + y_3 \cdot p_{10} \cdot Z_3 + y_3 \cdot p_{11} = 0$$

$$\vdots$$

$$\begin{bmatrix} -X_1 & -Y_1 & -Z_1 & -1 & 0 & 0 & 0 & 0 & x_1 \cdot X_1 & x_1 \cdot Y_1 & x_1 \cdot Z_1 & x_1 \\ 0 & 0 & 0 & 0 & -X_1 & -Y_1 & -Z_1 & -1 & y_1 \cdot X_1 & y_1 \cdot Y_1 & y_1 \cdot Z_1 & y_1 \\ -X_2 & -Y_2 & -Z_2 & -1 & 0 & 0 & 0 & 0 & x_2 \cdot X_2 & x_2 \cdot Y_2 & x_2 \cdot Z_2 & x_2 \\ 0 & 0 & 0 & 0 & -X_2 & -Y_2 & -Z_2 & -1 & y_2 \cdot X_2 & y_2 \cdot Y_2 & y_2 \cdot Z_2 & y_2 \\ -X_3 & -Y_3 & -Z_3 & -1 & 0 & 0 & 0 & 0 & x_3 \cdot X_3 & x_3 \cdot Y_3 & x_3 \cdot Z_3 & x_3 \\ 0 & 0 & 0 & 0 & -X_3 & -Y_3 & -Z_3 & -1 & y_3 \cdot X_3 & y_3 \cdot Y_3 & y_3 \cdot Z_3 & y_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \end{bmatrix} = 0$$

The System of Linear Equations for Camera Calibration

$$
\begin{bmatrix}
-X_1 & -Y_1 & -Z_1 & -1 & 0 & 0 & 0 & 0 & x_1 \cdot X_1 & x_1 \cdot Y_1 & x_1 \cdot Z_1 & x_1 \\
0 & 0 & 0 & 0 & -X_1 & -Y_1 & -Z_1 & -1 & y_1 \cdot X_1 & y_1 \cdot Y_1 & y_1 \cdot Z_1 & y_1 \\
-X_2 & -Y_2 & -Z_2 & -1 & 0 & 0 & 0 & 0 & x_2 \cdot X_2 & x_2 \cdot Y_2 & x_2 \cdot Z_2 & x_2 \\
0 & 0 & 0 & 0 & -X_2 & -Y_2 & -Z_2 & -1 & y_2 \cdot X_2 & y_2 \cdot Y_2 & y_2 \cdot Z_2 & y_2 \\
-X_3 & -Y_3 & -Z_3 & -1 & 0 & 0 & 0 & 0 & x_3 \cdot X_3 & x_3 \cdot Y_3 & x_3 \cdot Z_3 & x_3 \\
0 & 0 & 0 & 0 & -X_3 & -Y_3 & -Z_3 & -1 & y_3 \cdot X_3 & y_3 \cdot Y_3 & y_3 \cdot Z_3 & y_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \end{bmatrix} = 0
$$

obtained from 3D-2D correspondences

unknown

$$\mathrm{Q} \cdot \mathrm{M} = 0$$

Applying the Direct Linear Transform (DLT) Algorithm

$$\mathrm{Q} \cdot \mathrm{M} = 0$$

Minimal solution

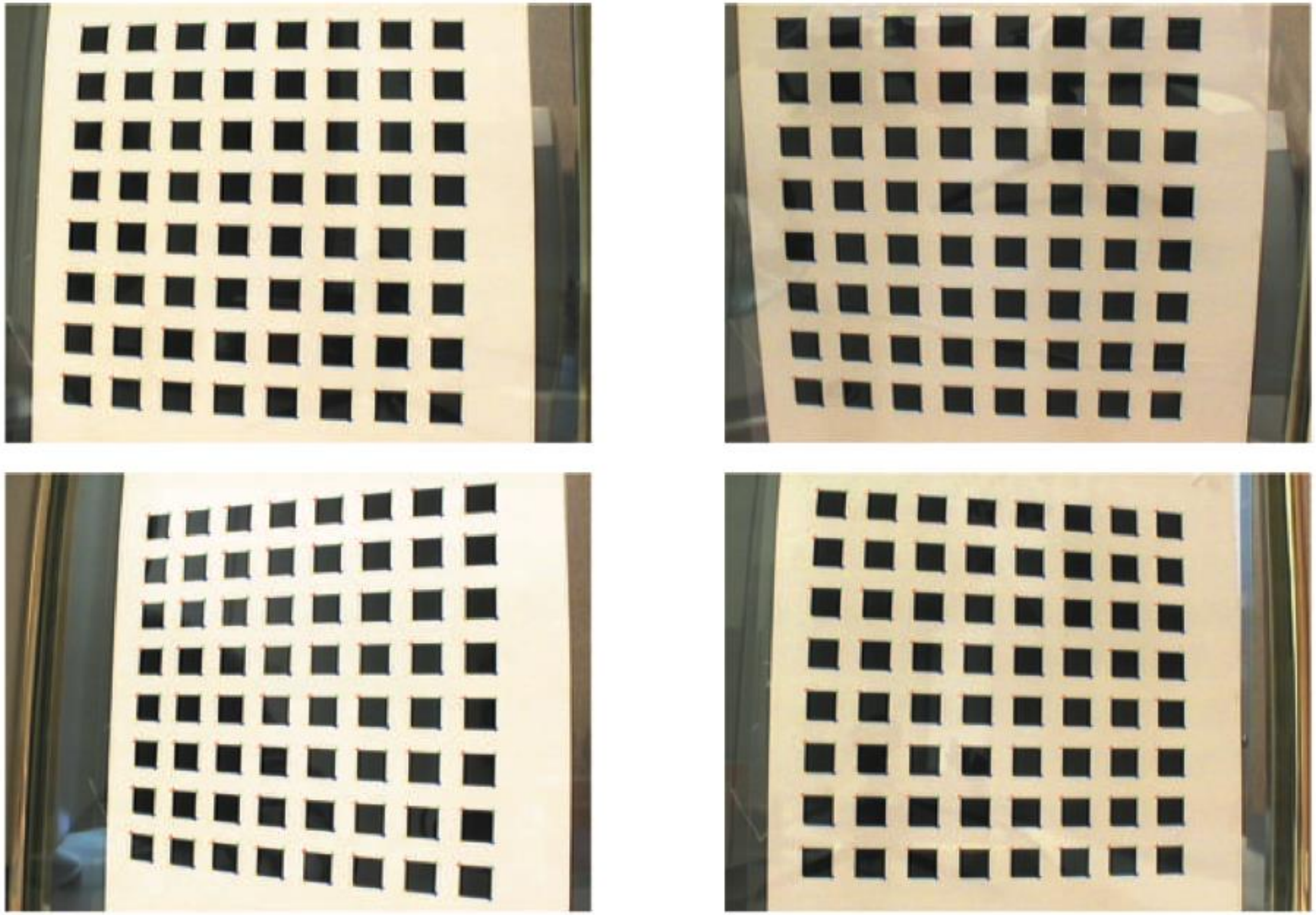
- $Q_{(2n \times 12)}$ should have rank 11 to have a unique (up to a scale) *non-zero* solution M
- Because each 3D-to-2D point correspondence provides 2 independent equations, then $5+\frac{1}{2}$ point correspondences are needed (in practice **6 point** correspondences!)

Over-determined solution

- For $n \geq 6$ points, a solution is the **Least Square solution**, which minimizes the sum of squared residuals, $||QM||^2$, subject to the constraint $||M||^2 = 1$. It can be solved through Singular Value Decomposition (SVD). The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix $Q^T Q$ (because it is the unit vector x that minimizes $||Qx||^2 = x^T Q^T Qx$.

Credit: https://rpg.ifi.uzh.ch/docs/teaching/2022/03_camera_calibration.pdf

SOUTH DAKOTA STATE UNIVERSITY

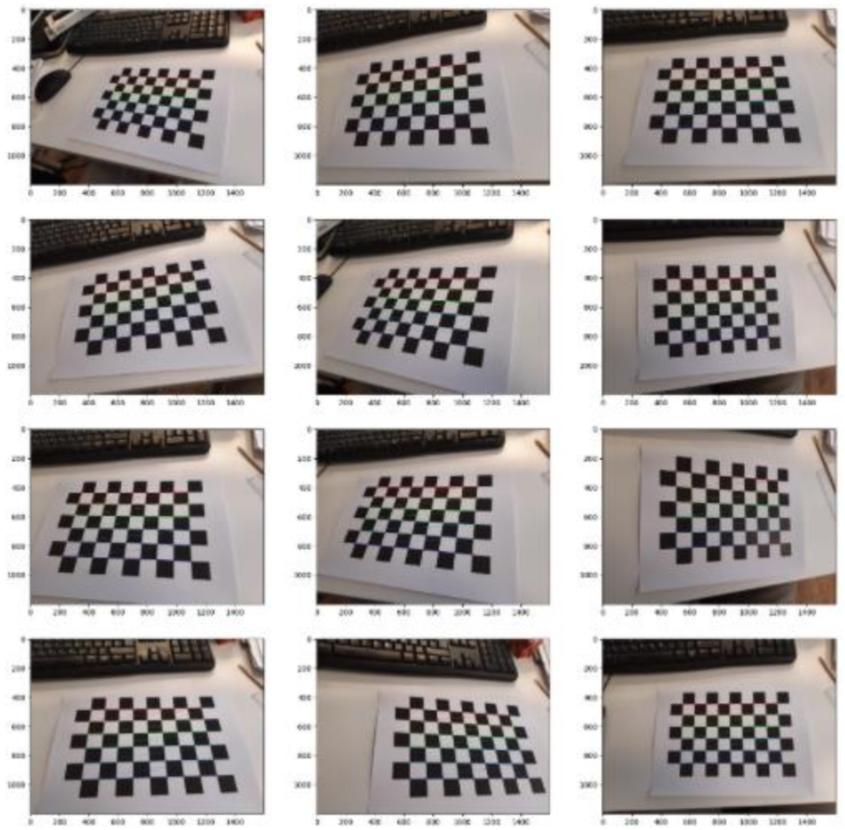


Zhang's Algorithm: Calibration from Planar Grids

Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.

Zhang's Algorithm: Calibration from Planar Grids

- Today's camera calibration toolboxes (e.g., OpenCV, Matlab) use multiple views of a planar grid (e.g., a checkerboard).
- They are based on a method developed in 2000 by Zhengyou Zhang at Microsoft Research.
- We have seen the OpenCV implementation in the Colab demo.



Zhang's Algorithm: Calibration from Planar Grids

- Similar to previous computation, but since the 3D points are all coplanar, we can let Z = 0, and thus we get

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

- We can write it as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

Zhang's Algorithm: Calibration from Planar Grids

- Again, from *n* points from a single view, we can get:

$$Q \cdot H = \mathbf{0}$$

where Q is known (based on the 3D-2D correspondences), H is unknown and to be solved for.

Minimal solution

- $Q_{(2n\times9)}$ should have rank 8 to have a unique (up to a scale) non-trivial solution H
- Each point correspondence provides 2 independent equations
- Thus, a minimum of **4 non-collinear points** is required

Solution for $\boldsymbol{n} \geq \mathbf{4}$ points

- It can be solved through Singular Value Decomposition (SVD) (same considerations as before)

Credit: https://rpg.ifi.uzh.ch/docs/teaching/2022/03_camera_calibration.pdf

How to Recover Intrinsic and Extrinsic Matrices from multiple views?

- Notice that each view j has a different homography H (and so a different rotation R and translation t), but K is the same for the all views.

$$\begin{bmatrix} h_{11}^{j} & h_{12}^{j} & h_{13}^{j} \\ h_{21}^{j} & h_{22}^{j} & h_{23}^{j} \\ h_{31}^{j} & h_{33}^{j} & h_{33}^{j} \end{bmatrix} = \begin{bmatrix} \mathrm{f_x} & 0 & u_0 \\ 0 & \mathrm{f_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11}^{j} & r_{12}^{j} & t_{1}^{j} \\ r_{21}^{j} & r_{22}^{j} & t_{2}^{j} \\ r_{31}^{j} & r_{32}^{j} & t_{3}^{j} \end{bmatrix}$$

Credit: https://rpg.ifi.uzh.ch/docs/teaching/2022/03_camera_calibration.pdf

How to Recover Intrinsic and Extrinsic Matrices from multiple views?

Won't be asked at the exam ☺

1. Estimate the homography H_i for each i-th view using the DLT algorithm.

2. Determine the intrinsics K of the camera from a set of homographies:
    1. Each homography $H_i \sim K(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t})$ provides two *linear* equations in the 6 entries of the matrix $\mathrm{B} := K^{-\top}K^{-1}$. Letting $\boldsymbol{w}_1 := K\boldsymbol{r}_1$, $\boldsymbol{w}_2 := K\boldsymbol{r}_2$, the rotation constraints $\boldsymbol{r}_1^\top\boldsymbol{r}_1 = \boldsymbol{r}_2^\top\boldsymbol{r}_2 = 1$ and $\boldsymbol{r}_1^\top\boldsymbol{r}_2 = 0$ become $\boldsymbol{w}_1^\top B\boldsymbol{w}_1 - \boldsymbol{w}_2^\top B\boldsymbol{w}_2 = 0$ and $\boldsymbol{w}_1^\top B\boldsymbol{w}_2 = 0$.
    2. Stack $2N$ equations from N views, to yield a linear system $A\boldsymbol{b} = \boldsymbol{0}$. Solve for $\boldsymbol{b}$ (i.e., B) using the Singular Value Decomposition (SVD).
    3. Use Cholesky decomposition to obtain K from B.

3. The extrinsic parameters for each view can be computed using K:
$\boldsymbol{r}_1 \sim \lambda K^{-1}H_i(:,1)$, $\boldsymbol{r}_2 \sim \lambda K^{-1}H_i(:,2)$, $\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$ and $T_i = \lambda K^{-1}H_i(:,3)$, with $\lambda = 1/K^{-1}H_i(:,1)$.
Finally, build $R_i = (\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ and enforce rotation matrix constraints.

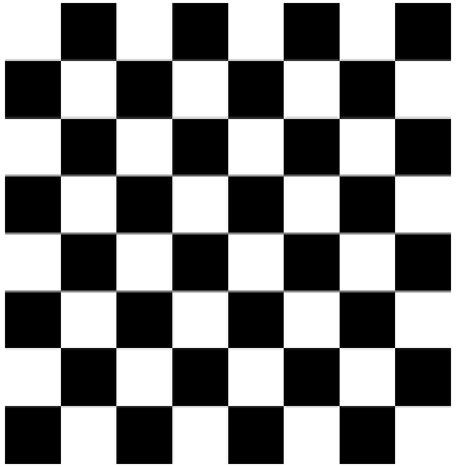
Credit: https://rpg.ifi.uzh.ch/docs/teaching/2022/03_camera_calibration.pdf

Reference: https://www.mathworks.com/help/vision/ug/camera-calibration.html

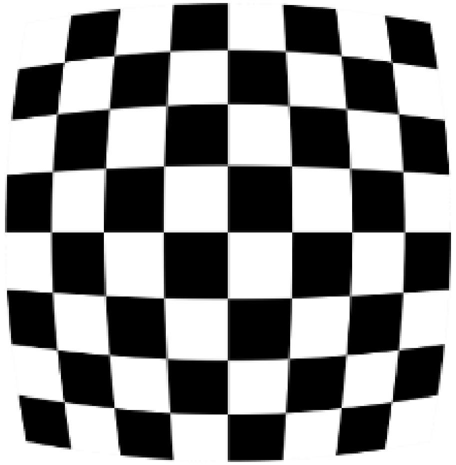
Camera Distortion

Credits: some slides of this section are adapted from CSC 492/592 (Spring 2024)

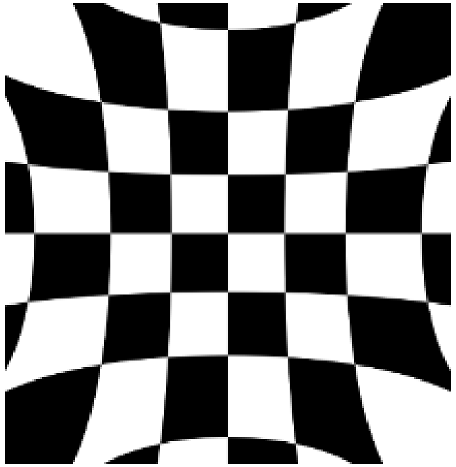
Lens Distortions:
Radial distortion



No distortion



Positive radial distortion
(Barrel distortion)



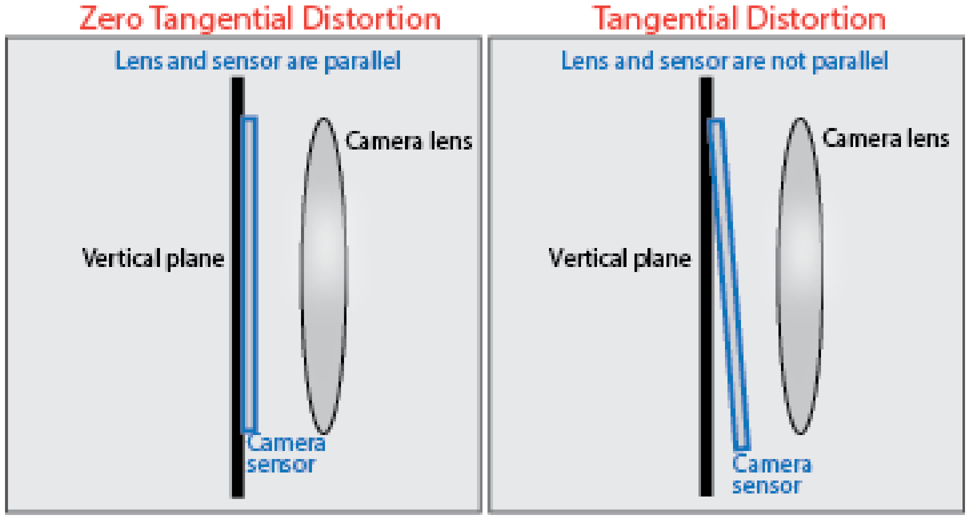
Negative radial distortion
(Pincushion distortion)

The radial distortion coefficients model this type of distortion:

$$x_{dist} = x(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$
$$y_{dist} = y(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$

'https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Lens Distortions: Tangential distortion



The tangential distortion coefficients model this type of distortion:

$$x_{dist} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$$
$$y_{dist} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$$

‘https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Let’s recap the OpenCV implementation!

```
double cv::calibrateCamera ( InputArrayOfArrays    objectPoints,
                             InputArrayOfArrays    imagePoints,
                             Size                  imageSize,
                             InputOutputArray      cameraMatrix,
                             InputOutputArray      distCoeffs,
                             OutputArrayOfArrays   rvecs,
                             OutputArrayOfArrays   tvecs,
                             OutputArray           stdDeviationsIntrinsics,
                             OutputArray           stdDeviationsExtrinsics,
                             OutputArray           perViewErrors,
                             int                   flags = 0,
                             TermCriteria          criteria = TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON) )
```

Python:

```
cv.calibrateCamera(    objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]]
```

Source: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

Let’s recap the OpenCV implementation!

```
double cv::calibrateCamera ( InputArrayOfArrays    objectPoints,
                             InputArrayOfArrays    imagePoints,
                             Size                  imageSize,
                             InputOutputArray      cameraMatrix,
                             InputOutputArray      distCoeffs,
                             OutputArrayOfArrays   rvecs,
                             OutputArrayOfArrays   tvecs,
                             OutputArray           stdDeviationsIntrinsics,
                             OutputArray           stdDeviationsExtrinsics,
                             OutputArray           perViewErrors,
                             int                   flags = 0,
                             TermCriteria          criteria = TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON) )
```

Python:

```
cv.calibrateCamera(    objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]]
```

Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern. Return camera distortion parameters as well.

Source: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

Takeaway

- Projection matrix *P*
- Google Colab
- Camera calibration using multiple views of a planar grid (e.g., a checkerboard)