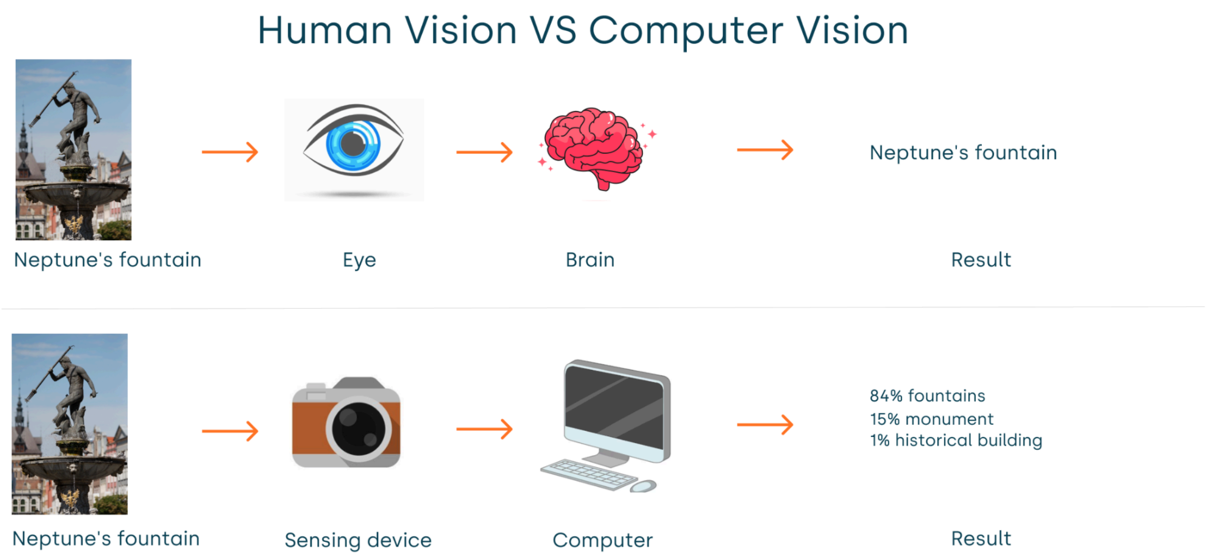
CHAPTER 1: INTRODUCTION TO COMPUTER VISION

1.1 Overview of Computer Vision

Computer vision is a sector of Artificial Intelligence that uses Machine Learning and Deep Learning to allow computers to see, recognize and analyze things in photos and videos in the same way that people do. Computational vision is rapidly gaining popularity for automated AI vision inspection, remote monitoring, and automation.
Definition:
Computer vision can be defined as a scientific field that extracts information out of digital images. The type of information gained from an image can vary from identification, space measurements for navigation, or augmented reality applications.



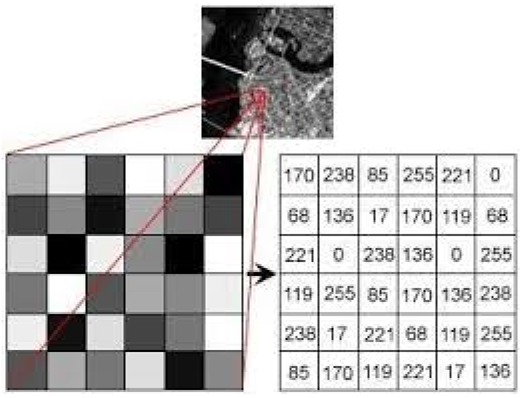
Computer vision systems use
(1) cameras to obtain visual data,
(2) machine learning models for processing the images, and
(3) conditional logic to automate application-specific use cases.
Applications
● Special effects Shape and motion capture are new techniques used in movies like Avatar to animate digital characters by recording the movements played by a human actor. In order to do that, we have to find the exact positions of markers on the actor's
face in a 3D space, and then recreate them on the digital avatar.
● 3D urban modeling Taking pictures with a drone over a city can be used to render a 3D model of the city. Computer vision is used to combine all the photos into a single 3D model.
● Scene recognition It is possible to recognize the location where a photo was taken. For instance, a photo of a landmark can be compared to billions of photos on google to find the best matches.

● Face detection Face detection has been used for multiple years in cameras to take better pictures and focus on the faces. Smile detection can allow a camera to take pictures automatically when the subject is smiling.
● Face recognition is more difficult than face detection, but with the scale of today's data, companies like Facebook are able to get very good performance. Finally, we can also use computer vision for biometrics, using unique iris pattern recognition or fingerprints.
● Optical Character Recognition One of the oldest successful applications of computer vision is to recognize characters and numbers. This can be used to read zip codes, or license plates.
● Mobile visual search With computer vision, we can do a search on Google using an image as the query.
● Self-driving cars Autonomous driving is one of the hottest applications of computer vision. Companies like Tesla, Google or General Motors compete to be the first to build a fully autonomous car.
● Automatic checkout Amazon Go is a new kind of store that has no checkout. With computer vision, algorithms detect exactly which products you take and they charge you as you walk out of the store
● Vision-based interaction Microsoft's Kinect captures movement in real time and allows players to interact directly with a game through moves.
● Augmented Reality AR is also a very hot field right now, and multiple companies are competing to provide the best mobile AR platform.Apple released ARKit in June and has already impressive applications
● Virtual Reality VR is using similar computer vision techniques as AR. The algorithm needs to know the position of a user, and the positions of all the objects around. As the user moves around, everything needs to be updated in a realistic and smooth way.

1.2 Basics of Image Representation

After getting an image, it is important to devise ways to represent the image. There are various ways by which an image can be represented. Let's look at the most common ways to represent an image.

Image as a matrix



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

The simplest way to represent the image is in the form of a matrix.
In fig., a part of the image, i.e., the clock, has been represented as a matrix. A similar matrix\will represent the rest of the image too.It is commonly seen that people use up to a byte to represent

every pixel of the image. This means that values between 0 to 255 represent the intensity for each pixel in the image where 0 is black and 255 is white. For every color channelin the image,one such matrix is generated. In practice, it is also common to normalize the values between 0 and 1 (as done in the example in the figure above).

Image as a function

An image can also be represented as a function. An image (grayscale) can be thought of as a function that takes in a pixel coordinate and gives the intensity at that pixel.
It can be written as function

f: R2 → R that outputs the intensity at any input point (x,y). The
value of intensity can be between 0 to 255 or 0 to 1 if values are normalized.

Numericals:

Resolution Conversion Example:

1. Given an image with a resolution of 1024x768 and pixel depth of 24 bits, calculate the size of the image in memory.

   Solution: Image size (in bits)=Width×Height×Color depth =1024×768×24=18,874,368 bits=2.25 MB(to convert into MB divide bits with 1024*1024*8)

2. Calculate the total number of pixels in an RGB image with a resolution of 1920×1080. How many bits are required to store this image assuming an 8-bit depth per channel?

   Solution:
   The number of pixels in the image is:
   1920×1080=2,073,600
   1920×1080=2,073,600 pixels.

   Since it is an RGB image, each pixel has three color channels (Red, Green, Blue). The total number of values is:
   2,073,600×3=6,220,800 values.

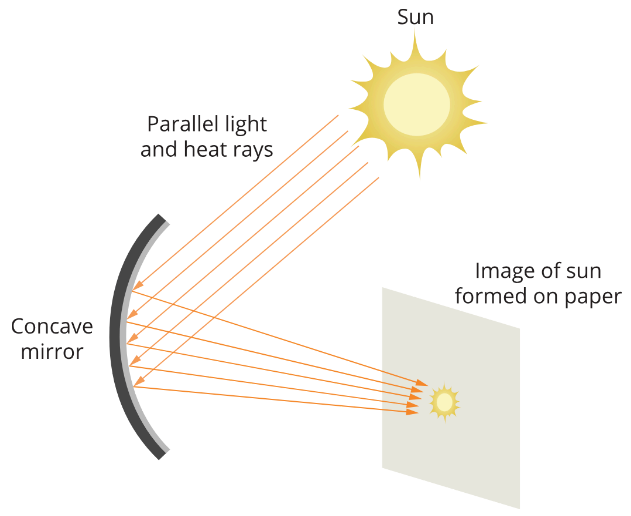
   With an 8-bit depth for each channel, the total number of bits required to store the image is:
   6,220,800×8=49,766,400 bits.
   To convert to bytes:
   49,766,400÷8=6,220,800 bytes.

1.3 Image Formation

In image formation,radiometry is concerned with the relation among the amounts of light energy emitted from light sources,reflected from surfaces and captured by sensors.
Simple model for Image Formation
● A Simple model of image formation The scene is illuminated by a single source.

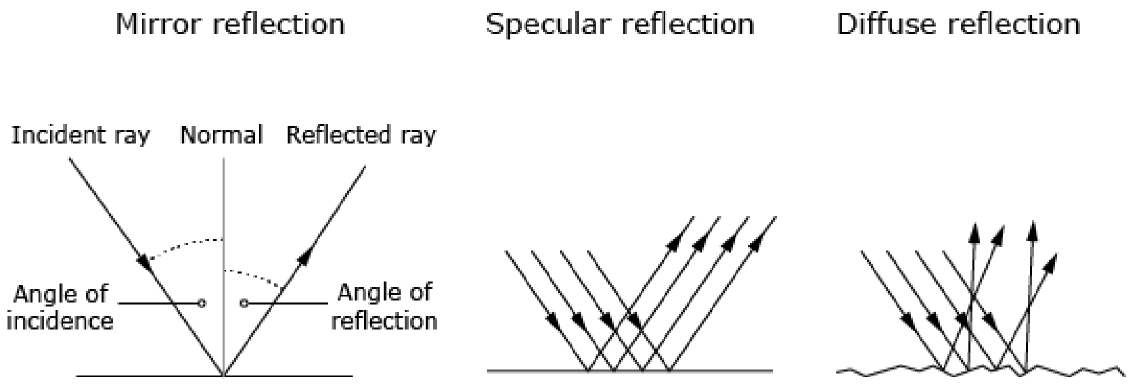
● The scene reflects radiation towards the camera. The camera senses it via solid state cells (CCD cameras)
● There are two parts to the image formation process:
○ The geometry, which determines where in the image plane the projection of a point in the scene will be located.
○ The physics of light, which determines the brightness of a point in the image plane. $f(x,y) = i(x,y)\ r(x,y)$
■ Simple model: i: illumination, r: reflectance
Photometric image formation

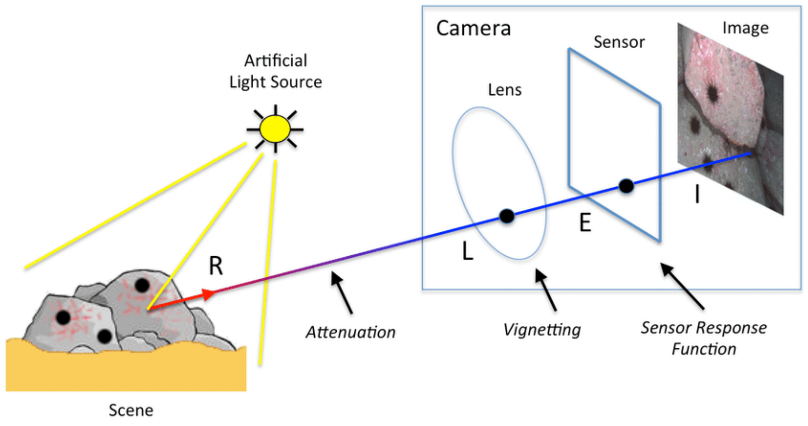
● Images cannot exist without light.
● Light sources can be a point or an area light source.
  ○ point source (location only, e.g. bulb)
  ○ Directional source (orientation only, e.g. Sun)
  ○ Ambient source (no location nor orientation)
  ○ Spot light (point source + spread angle)
  ○ Flap, barn-door (directional source + spatial extent)
● When Light arriving at a surface, two factors affect of light for vision.
  ○ Strength: characterized by its irradiance (energy/time-area)
  ○ Distance: how much emitted energy actually gets to the object (no attenuation, no reflection)



● When the light hits a surface, three major reactions might occur-

  ○ Some light is absorbed. That depends on the factor called ρ (albedo). Low ρ of the surface means more light will get absorbed.
  ○ Some light gets reflected diffusively, which is independent of viewing direction. E.g., cloth, brick.
  Some light is reflected specularly, which depends on the viewing direction. E.g., mirror.
  ○ Some lights may refracted.(absorbed and travel through material)
  ○ absorption + reflection + refraction = total incident





1. Pinhole Camera Model

Example Question: Given a pinhole camera model with a focal length of 50 mm, and the distance between the object and the camera is 500 mm, calculate the size of the image of an object that is 100 mm tall.

Solution: Using the formula for image height in the pinhole camera model:

$$\frac{\text{Image Height}}{\text{Object Height}} = \frac{\text{Focal Length}}{\text{Object Distance}}$$

$$\frac{\text{Image Height}}{100} = \frac{50}{500}$$

$$\text{Image Height} = \frac{50 \times 100}{500} = 10 \text{ mm}$$

So, the image of the object will be 10 mm tall.

1.4 Camera Calibration:

https://alphapixeldev.com/opencv-tutorial-part-1-camera-calibration/
https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html
https://learnopencv.com/camera-calibration-using-opencv/

What is Camera calibration?

Camera calibration is a crucial step in most computer vision tasks, as it allows you to determine the intrinsic parameters of a camera, which are essential for accurately interpreting the geometry of the scene being captured. Camera calibration involves estimating the camera's internal parameters, such as the focal length, optical center (principal point), and lens distortion coefficients. These parameters describe how the camera projects 3D points in the world onto a 2D image plane. The goal of camera calibration is to correct distortions and improve the accuracy of measurements and image processing tasks, such as 3D reconstruction, object detection, and augmented reality applications.

- The process of estimating the parameters of a camera is called camera calibration.
- This means we have all the information (parameters or coefficients) about the camera required to determine an accurate relationship between a 3D point in the real world and its corresponding 2D projection (pixel) in the image captured by that calibrated camera.

Two kinds of parameters

1. Internal parameters of the camera/lens system. E.g. focal length, optical center, and radial distortion coefficients of the lens.
2. External parameters : This refers to the orientation (rotation and translation) of the camera with respect to some world coordinate system.

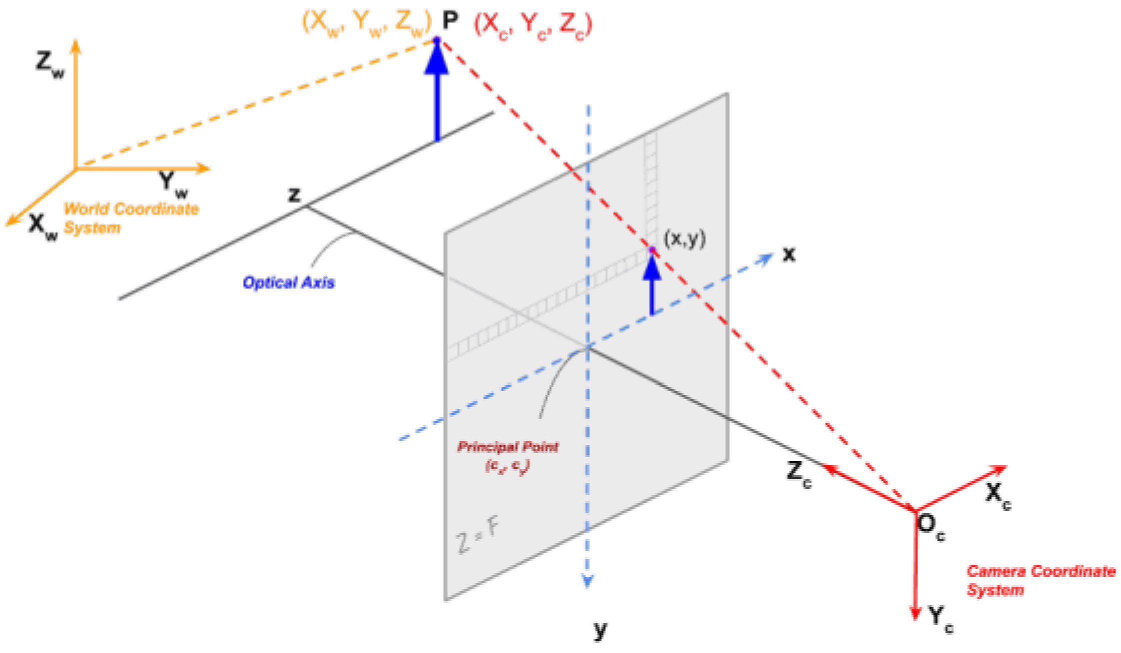
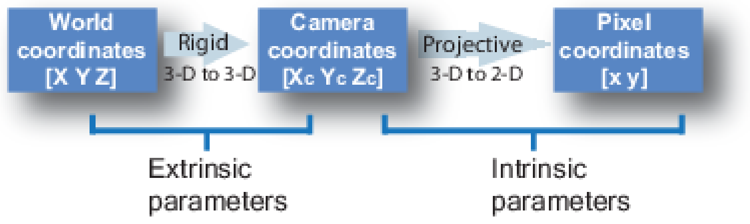


Figure 2: The projection of point P onto the image plane is shown.

The major purpose of camera calibration is to remove the distortions in the image and thereby establish a relation between image pixels and real world dimensions. In order to remove the distortion we need to find the intrinsic parameters in the intrinsic matrix K and the distortion parameters.
Intrinsic parameters depend only on camera characteristics while extrinsic parameters depend on camera position.
Extrinsic Parameters:
Mapping of 3D world coordinate to 2D Image coordinate
Calibration maps a 3D point (in the world) with [X, Y, Z] coordinates to a 2D Pixel with [X, Y] coordinates.
With a calibrated camera, we can transformation world coordinates to pixel coordinates going through camera coordinates.



$$o_{camera} = [R|t]O_{world}$$

Extrinsic Calibration Formula

The essential matrix is the part of the basic fundamental matrix that is only related to external parameters.
Extrinsic calibration converts World Coordinates to Camera Coordinates. The extrinsic parameters are called R (rotation matrix) and T (translation matrix).

Intrinsic Parameters
Intrinsic calibration converts Camera Coordinates to Pixel Coordinates. It requires inner values for the camera such as focal length, optical center. The intrinsic parameter is a matrix we call K.

Camera to Image Conversion

$$o_{image} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} o_{camera} = K o_{camera}$$

Intrinsic Calibration Formula

$$o_{image} = PO = K[R|t]O_{world}$$

World to Image Mapping Formula

However the matrices doesn't match. Due to this world matrix needs to be modified from [X Y Z] to [X Y Z 1]. This "1" is called a homogeneous coordinate.

$$o_{image} = PO = \underbrace{\underset{3\text{x}3}{K}\;\underset{3\text{x}4}{[R|t]}}_{3\text{x}4}\;\underset{3\text{x}1}{O_{world}} \longrightarrow \begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

Using Homogeneous Coordinate in World Matrix

$$P = K[R|T]$$

If we have the 2D coordinates, then using calibration parameters, we can map to 3D and vice versa using the following equation:

$$\begin{pmatrix}u\\v\\1\end{pmatrix} \sim \begin{pmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{pmatrix}\begin{pmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix}$$

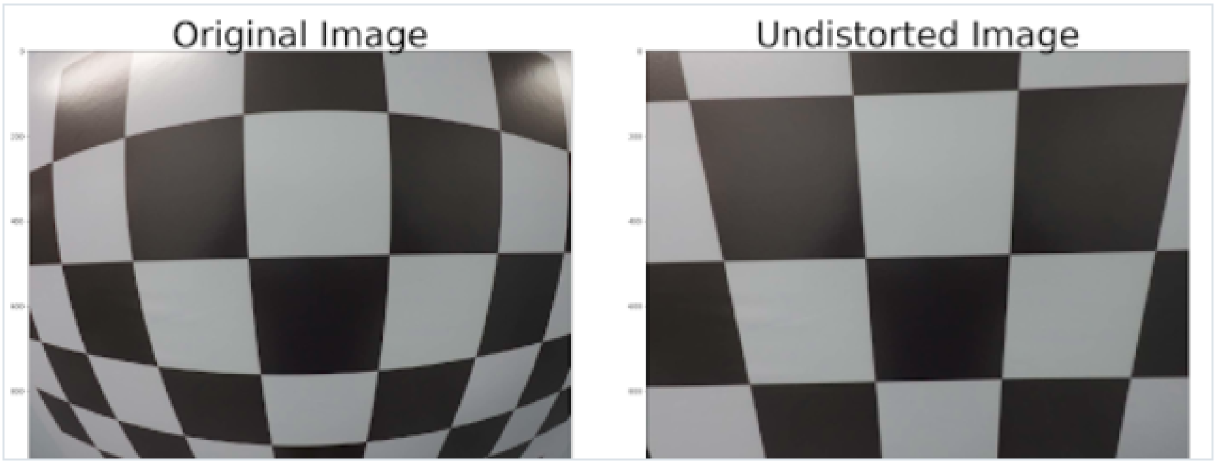
Pinhole Camera Model

$$\begin{pmatrix}u\\v\\1\end{pmatrix} = K * M_{ex} * \begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}$$

Simplified Pinhole Camera Model

Image Distortion

A distortion can be radial or tangential. Calibration helps to undistort an image.



Distorted and Undistorted image

Radial distortion: Radial distortion occurs when the light rays bend more at the edges of the lens than the optical center of the lens. It essentially makes straight lines appear as slightly curved within an image.

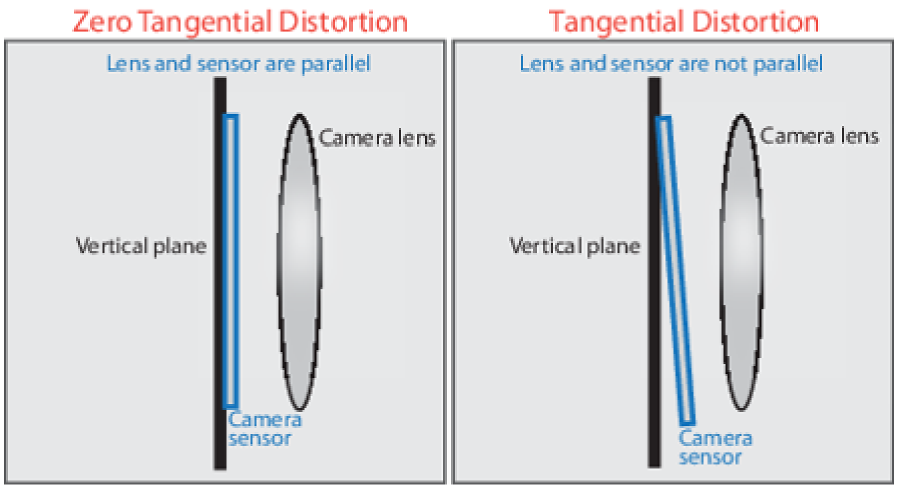
x-distorted $= x(1 + k1 * r^2 + k2 * r^4 + k3 * r^6)$

y-distorted $= y(1 + k1 * r^2 + k2 * r^4 + k3 * r^6)$

x, y — undistorted pixels that are in image coordinate system.

k1, k2, k3 — radial distortion coefficients of the lens.

Tangential distortion: This form of distortion occurs when the lens of the camera being utilized is not perfectly aligned i.e. parallel with the image plane. This makes the image to be extended a little while longer or tilted, it makes the objects appear farther away or even closer than they actually are.



x-distorted $= x + [2 * p1 * x * y + p2 * (r^2 + 2 * x^2)]$

y-distorted $= y + [p1 * (r^2 + 2 * y^2) + 2 * p2 * x * y]$

x, y — undistorted pixels that are in image coordinate system.

p1, p2 — tangential distortion coefficients of the lens.

This distortion can be captured by five numbers called Distortion Coefficients, whose values reflect the amount of radial and tangential distortion in an image.

$$\text{Distortion coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

We need intrinsic and extrinsic parameters of camera to find distortion coefficient parameters (k1, k2, k3, p1, p2). The intrinsic parameters are camera-specific (same parameters for same camera) that are focal length (fx, fy) and optical centers (cx, cy).

The actual OpenCV Camera Calibration step

- For each image
    - Load the image
    - Convert to Grayscale
    - Find the chessboard corners
    - Refine the location of the corners
    - Add the corners to the dataset
- Perform calibrateCamera()

```
# Import required modules
import cv2
import numpy as np
import os
import glob
from google.colab.patches import cv2_imshow

# Define the dimensions of checkerboard
CHECKERBOARD = (6, 9)



# stop the iteration when specified
# accuracy, epsilon, is reached or
# specified number of iterations are completed.
criteria = (cv2.TERM_CRITERIA_EPS +
      cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# Vector for 3D points
threedpoints = []

# Vector for 2D points
twodpoints = []
# 3D points real world coordinates
objectp3d = np.zeros((1, CHECKERBOARD[0]
          * CHECKERBOARD[1],
          3), np.float32)
objectp3d[0, :, :2] = np.mgrid[0:CHECKERBOARD[0],
              0:CHECKERBOARD[1]].T.reshape(-1, 2)
prev_img_shape = None
# Extracting path of individual image stored
# in a given directory. Since no path is
# specified, it will take current directory
# jpg files alone
images = glob.glob('/content/*.jpg')

for filename in images:
  image = cv2.imread(filename)
  grayColor = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

  # Find the chess board corners
  # If desired number of corners are
  # found in the image then ret = true
  ret, corners = cv2.findChessboardCorners(
          grayColor, CHECKERBOARD,
          cv2.CALIB_CB_ADAPTIVE_THRESH
          + cv2.CALIB_CB_FAST_CHECK +
          cv2.CALIB_CB_NORMALIZE_IMAGE)
  # If desired number of corners can be detected then,
  # refine the pixel coordinates and display
  # them on the images of checker board
  if ret == True:
    threedpoints.append(objectp3d)

    # Refining pixel coordinates
    # for given 2d points.
    corners2 = cv2.cornerSubPix(
      grayColor, corners, (11, 11), (-1, -1), criteria)

    twodpoints.append(corners2)

    # Draw and display the corners
    image = cv2.drawChessboardCorners(image,
                    CHECKERBOARD,
                    corners2, ret)
  cv2_imshow(image)
  cv2.waitKey(0)
cv2.destroyAllWindows()
h, w = image.shape[:2]

# Perform camera calibration by
# passing the value of above found out 3D points (threedpoints)
# and its corresponding pixel coordinates of the
# detected corners (twodpoints)
ret, matrix, distortion, r_vecs, t_vecs = cv2.calibrateCamera(
  threedpoints, twodpoints, grayColor.shape[::-1], None, None)

# Displaying required output
print(" Camera matrix:")
print(matrix)

print("\n Distortion coefficient:")
print(distortion)

print("\n Rotation Vectors:")
print(r_vecs)

print("\n Translation Vectors:")
print(t_vecs)
# Refining the camera matrix using parameters obtained by calibration
#newcameramtx, roi = cv2.getOptimalNewCameraMatrix(matrix, distortion,
(w,h), 1, (w,h))

# Method to undistort the image
#dst = cv2.undistort(graycolor, matrix, distortion, None, newcameramtx)

#dst = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)
 # Displaying the undistorted image
#cv2.imshow("undistorted image",dst)
#cv2.waitKey(0)
```

Examples:

A camera has intrinsic parameters: focal length (fx = 1000, fy = 950), and the optical center (cx = 320, cy = 240). If a 3D point is at (X = 2, Y = 1, Z = 5) in the world coordinate system, calculate its corresponding 2D image coordinates.

Solution:

The image coordinates are computed using the formula:

$$u = \frac{fx \times X}{Z} + cx$$

$$v = \frac{fy \times Y}{Z} + cy$$

Substituting the values:

$$u = \frac{1000 \times 2}{5} + 320 = 400 + 320 = 720$$

$$v = \frac{950 \times 1}{5} + 240 = 190 + 240 = 430$$

Thus, the 2D image coordinates are **(720, 430)**.

If a camera has a focal length of 50mm and an object is located 2000mm away from the camera, what is the size of the object's image on the sensor, assuming the actual object is 500mm tall?

Using the thin lens equation:

$$\text{Image size} = \frac{\text{Focal length}}{\text{Object distance}} \times \text{Object size}$$

Substituting the values:

$$\text{Image size} = \frac{50}{2000} \times 500 = 12.5\,\text{mm}$$

So, the size of the object's image on the sensor is **12.5mm**.

Example Question: A point in 3D space is given as $P = (200, 150, 500)$. If the focal length of the camera is 50 mm, find the 2D coordinates of the point after projection on the image plane using homogeneous coordinates.

Solution: For projection, we use:

$$x' = \frac{f \cdot X}{Z}, \quad y' = \frac{f \cdot Y}{Z}$$

Given that $X = 200$, $Y = 150$, $Z = 500$, and $f = 50$:

$$x' = \frac{50 \times 200}{500} = 20, \quad y' = \frac{50 \times 150}{500} = 15$$

So, the 2D projected coordinates are $(20, 15)$.