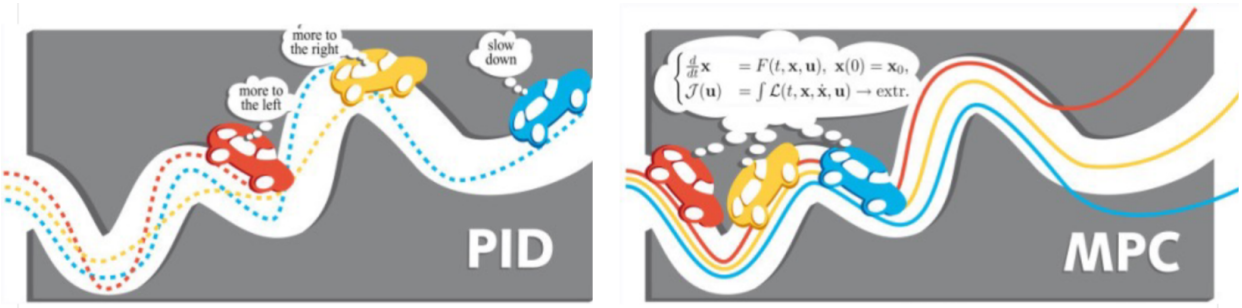
CS 188 Robotics Week 3

Aidan Jan

April 17, 2025

Model Predictive Control

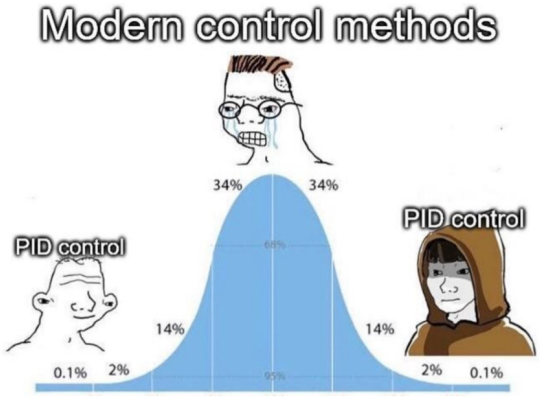


Model predictive control (MPC) is an optimal control technique in which the calculated control actions minimize a cost function for a constrained dynamical system over a finite, receding, horizon.

- Predicts future system behavior using a **model**
- Solves an **optimization** problem at each step.
- Applies only the first control input at each step (iterative).
- Repeats this process continuously (receding horizon).
- Handles input and output **constraints** directly.

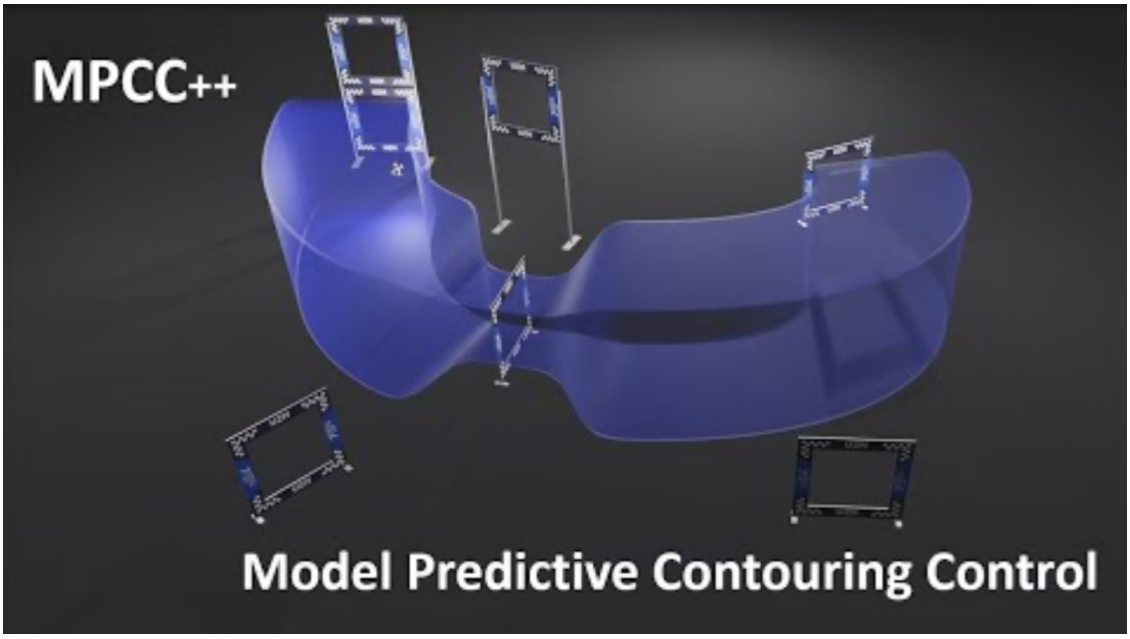
System Identification

. . . building a mathematical model of a dynamic system **from measured data**



"PID is used in 99% of applications and 99% of research efforts go into the remaining 1% that can't be solved with PID"

Model Predictive Contouring Control (MPCC++)



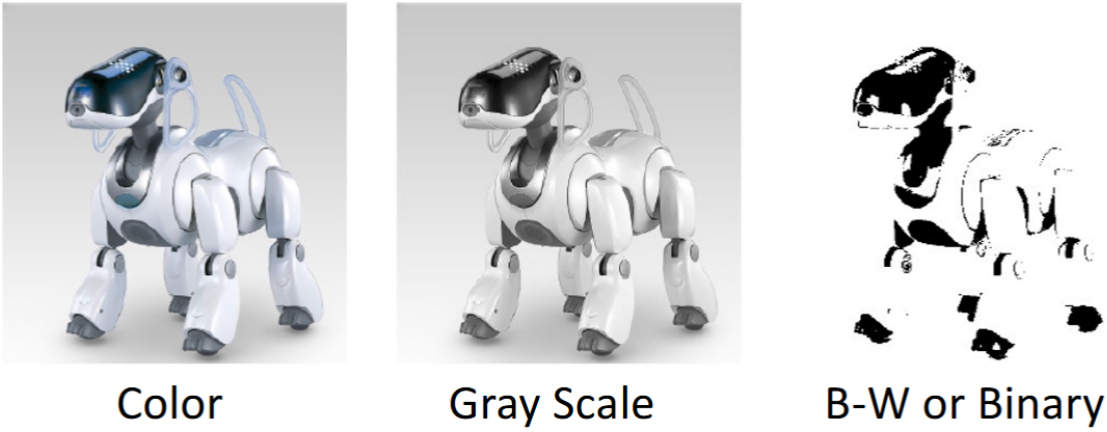
Cameras and 2D Perception

Color Camera

- Cameras are the primary sensor for many robotic platforms
- One of the cheapest and richest sensors is a camera
- Many other sensors are built on top of the color camera

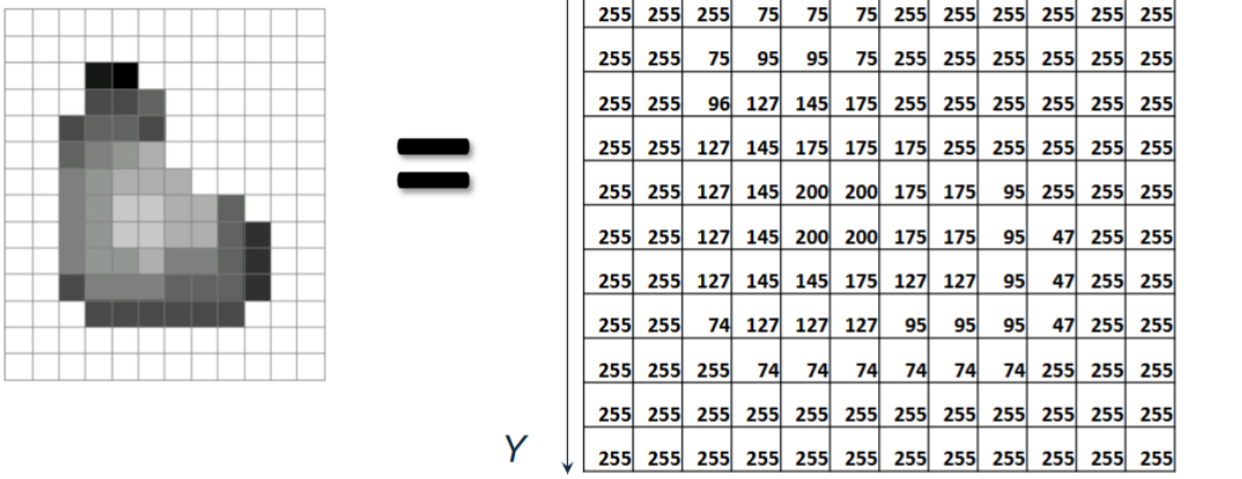
Images Representation

- An image is basically a 2D array of intensity/color values
- Image types:



Grayscale Images

A grid (2D matrix) of intensity values:

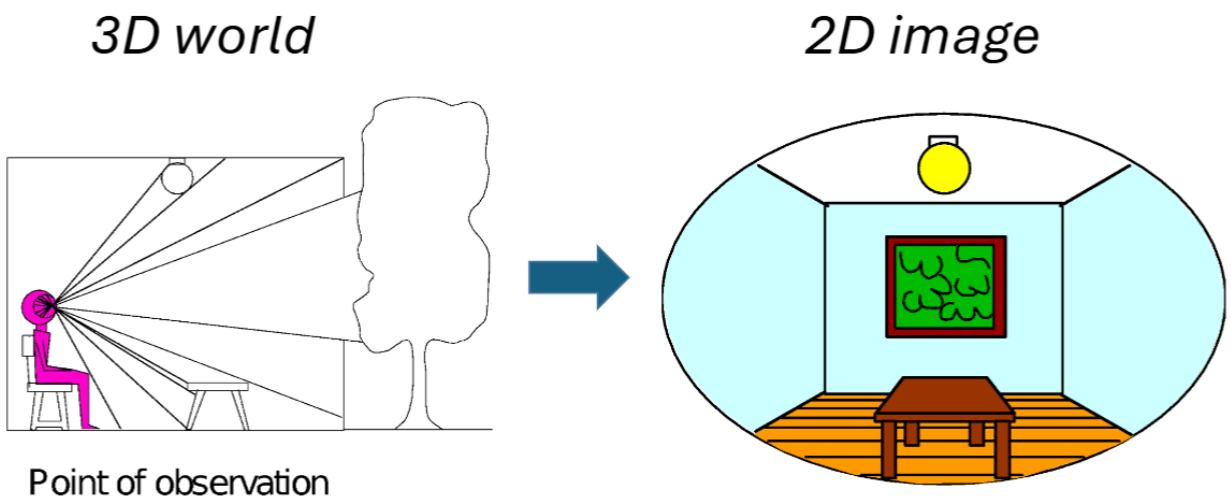


- Pixel: A "picture element" that contains the light intensity at some location (x, y) in the image Referred to as $I(x, y)$
- Image Resolution: expressed in terms of Width and Height of the image

Camera and Image Formation

- How we get the image (Image formation)
- Pinhole Camera Model
- 2D computer vision tasks and challenges

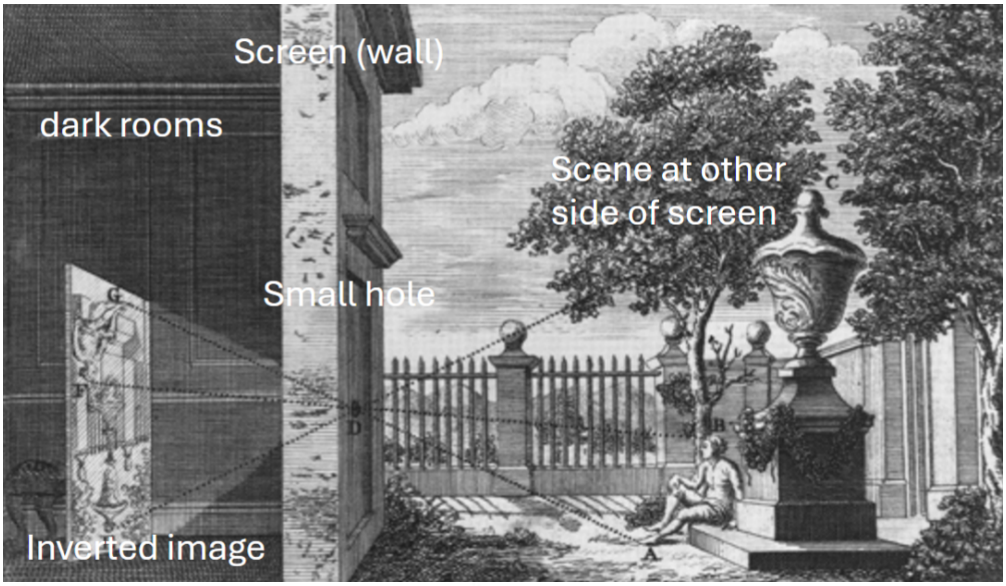
Image Formation

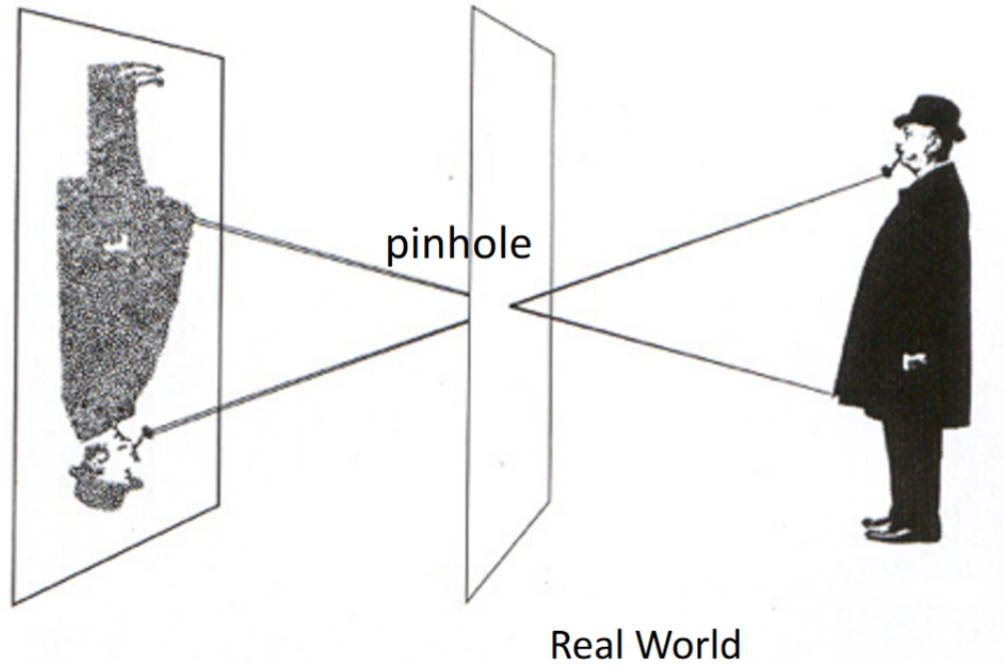


Pinhole Camera Model

Pinhole image: Natural phenomenon, known during classical period in China and Greece (e.g., 470 BCE to 390 BCE).

- Used for art creation and religious ceremony in the ancient times.
- Expensive to record the image (drawing)

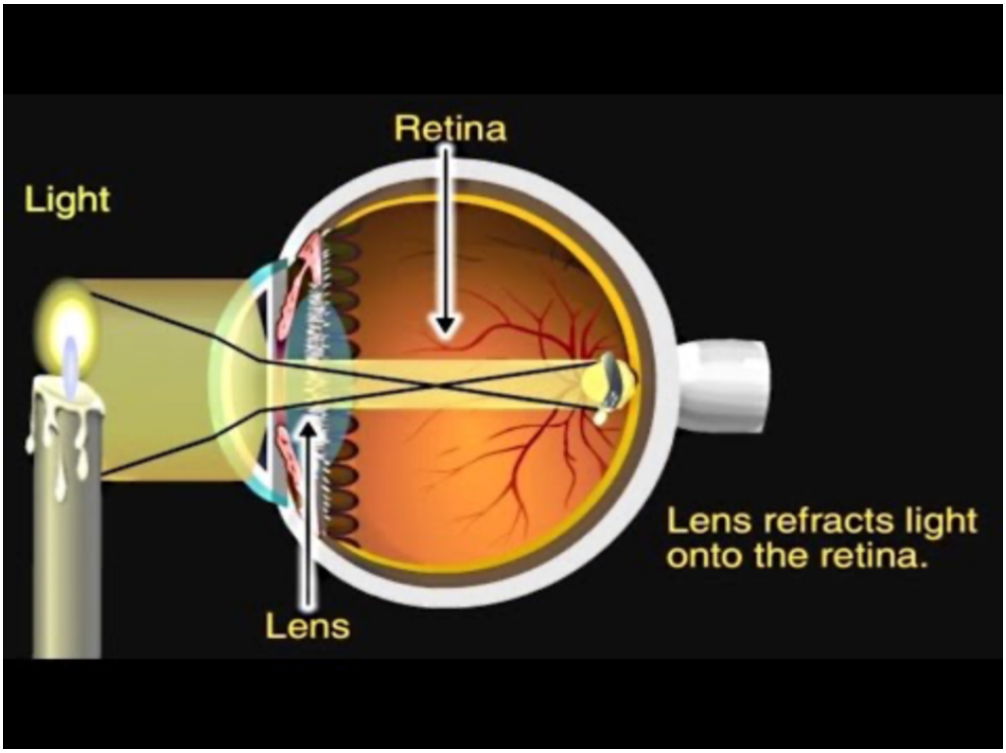




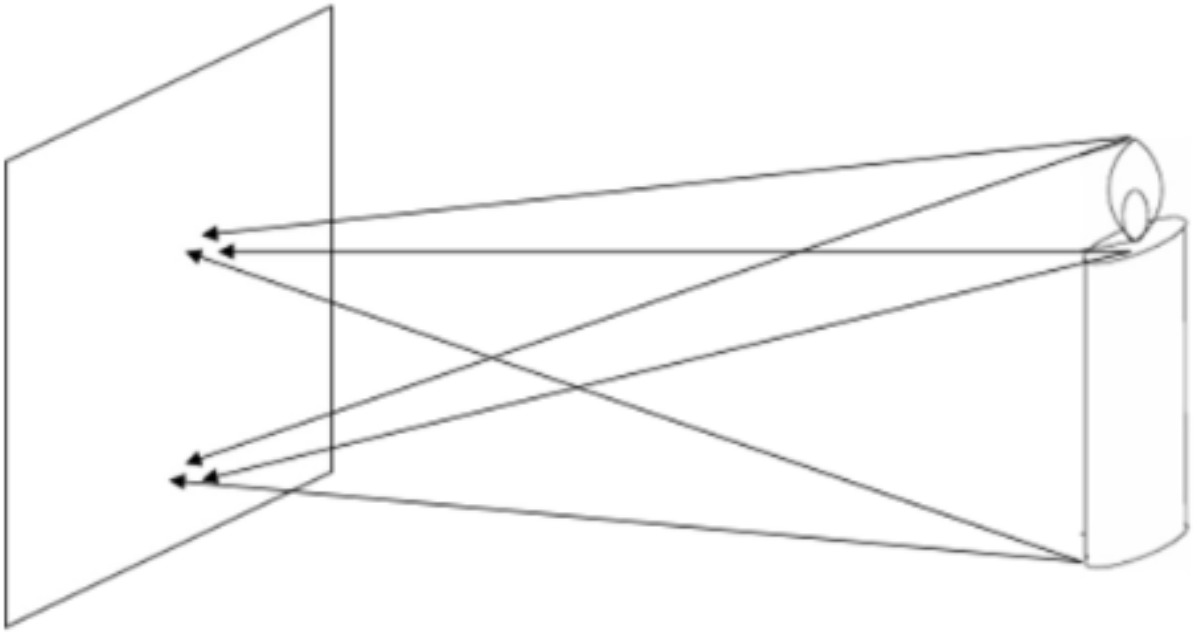
- Light sensitive material was used as film.
- Hard to store, loses color after a while

Today: photon sensors are CCD, CMOS, etc.

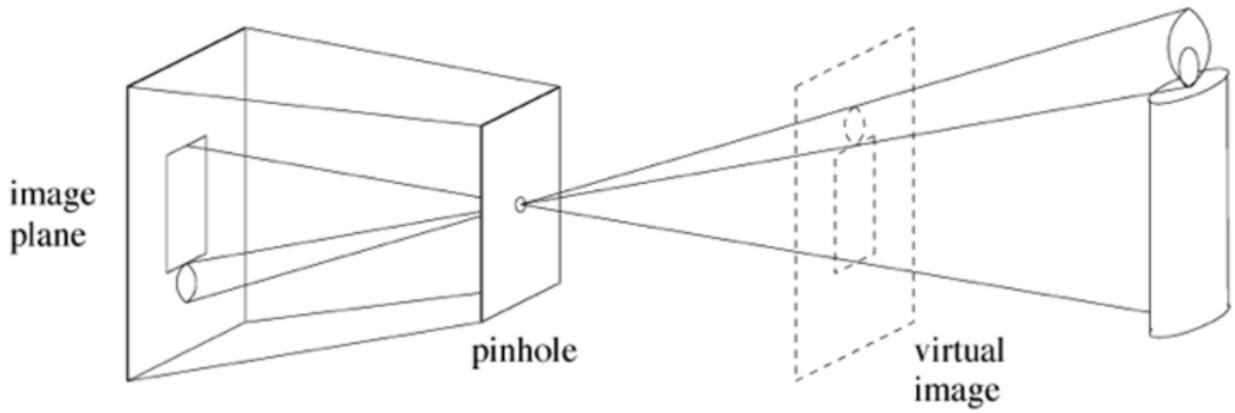
Human Vision



Why do we need a pinhole?



Light rays from many different parts of the scene strike the same point on the paper.



Each point on the image plane sees light from only one direction - the one that passes through the pinhole.

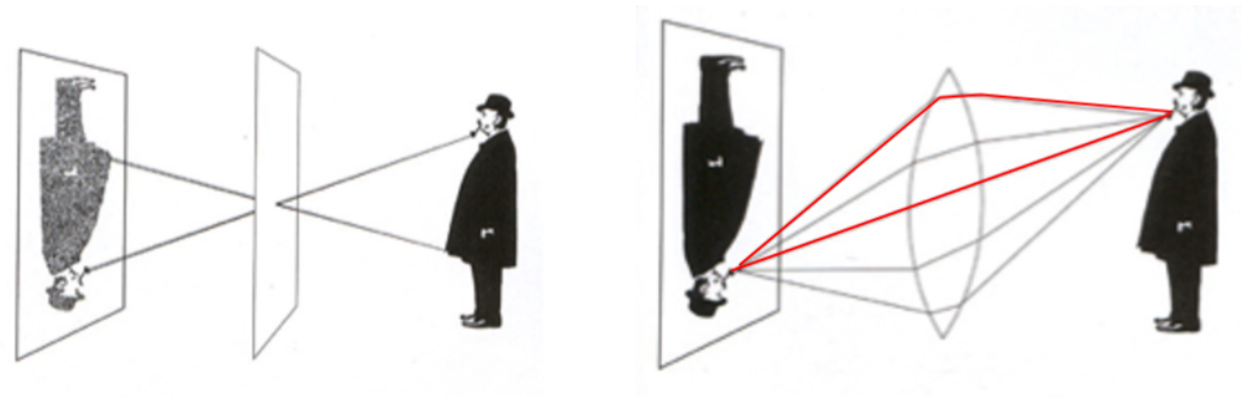
Problem with Pinhole Camera

The pinhole size:

- If large, blurry
- If small, not enough light
- When the pinhole size is extremely small, we will see the diffraction effect through the pinhole, resulting in the blurry image

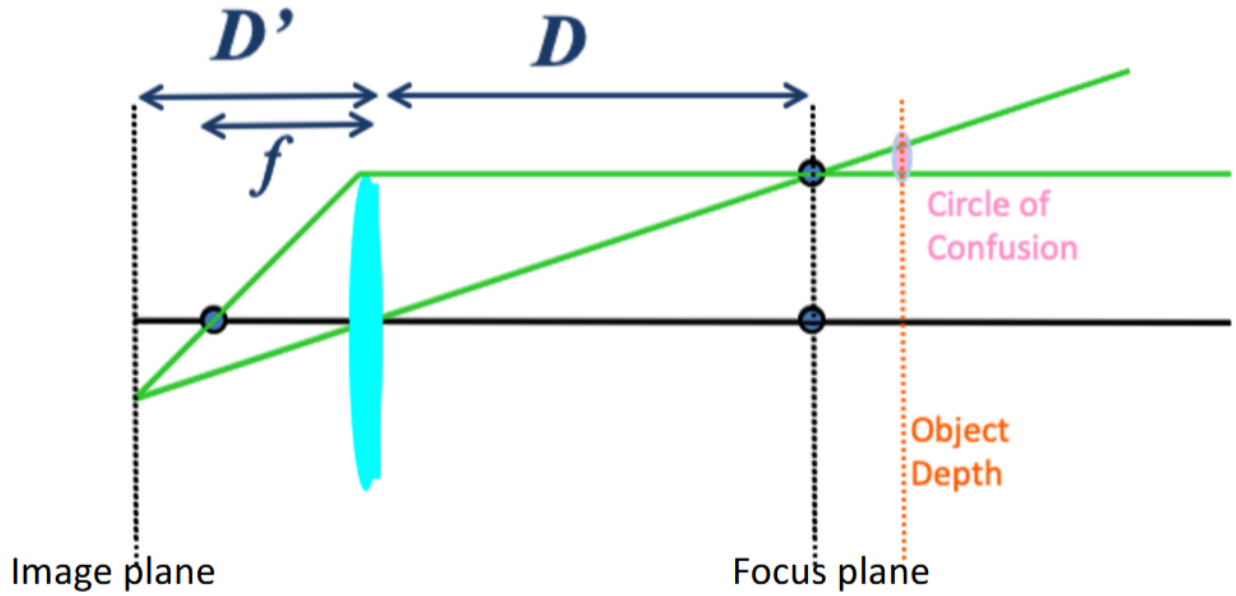
Solution: refraction (lenses)

- Essentially add multiple pinhole images
- Shift them to align using the light **refraction**
- However, this alignment works only for one depth (need the object and image plane to stay in focus.)



Lenses Issues (depth of field)

Only objects on focus plane are in "perfect" focus



$$\frac{1}{D} + \frac{1}{D'} = \frac{1}{f}$$

where D is the distance of a focus plane to the lens plane, D' is the distance of the image plane to the lens plane, and f is the focal length of the lens.

- Objects close to the focus plane are in better focus
- Objects further away are not.

Camera Terminology

These terms will be defined below.

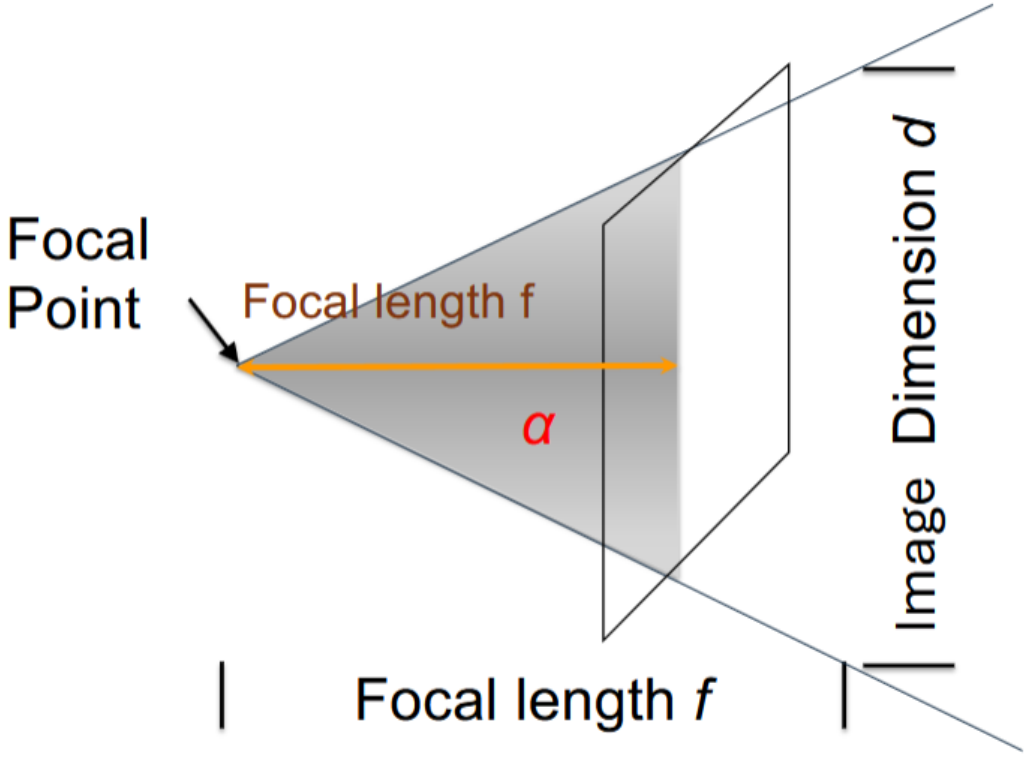
- Focal length
- Field of view
- Aperture
- Camera intrinsic
- Camera extrinsic

Pinhole Camera Geometry

Motivation

- Physics of real cameras are all different (too tedious to model all of them).
- But they all try their best to approximate pinhole camera.
- So **in most of computer vision subjects, we model all cameras mathematically as a pinhole camera.**

Field of View (FOV)

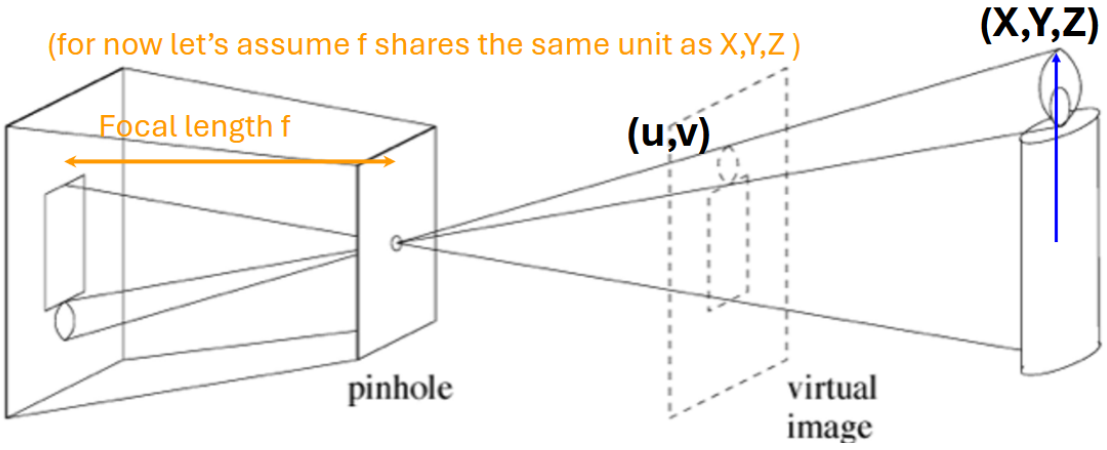


Distance from the focal point to image plane

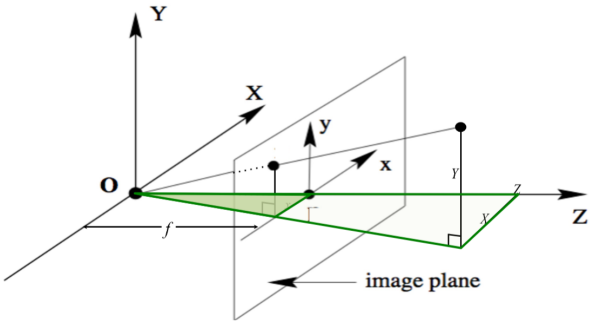
$$\alpha = 2 \arctan \frac{d}{2f}$$

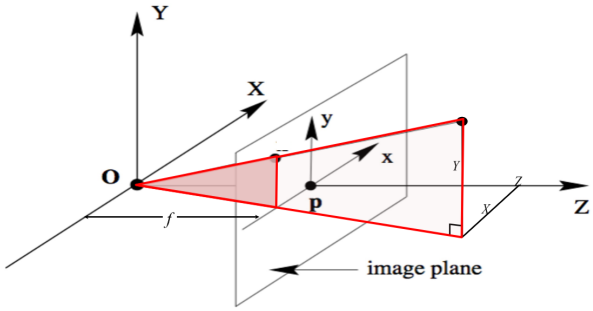
- The unit of FoV α is a degree.
- Each camera has two FoV: vertical and horizontal.

Focal Length



Camera Projection





$$\frac{u}{f} = \frac{X}{Z}$$
$$\frac{v}{f} = \frac{Y}{Z}$$

In camera coordinates, the camera center is the origin.

$$p_{2d} = \begin{bmatrix} u \\ v \end{bmatrix} \qquad p_{3d} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$u = \frac{fX}{Z}$$
$$v = \frac{fY}{Z}$$

$$\Leftrightarrow \quad \lambda \begin{bmatrix} u \\ v \\ f \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Cartesian Coordinate

$$(x, y) \Rightarrow$$

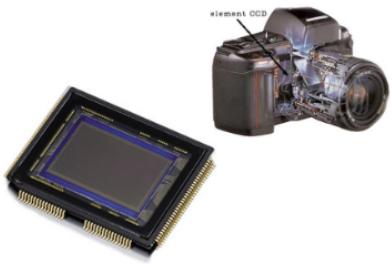
homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

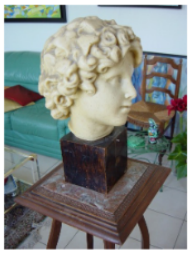
Equivalent
(w is non-zero scalar)

Convert from homogenous coordinate back to 2D coordinate

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \left(\frac{a}{c}, \frac{b}{c}\right)$$



If we let f to take care transform from word unit to image unit.
The unit of f is pixel



World Unit: e.g. Meters

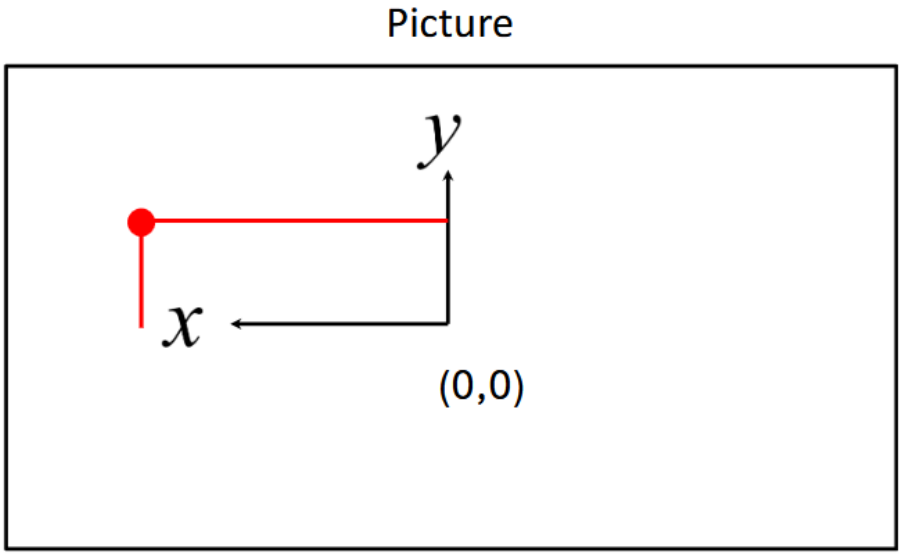
Image Unit: Pixels

$$u = \frac{fX}{Z} \qquad v = \frac{fY}{Z}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

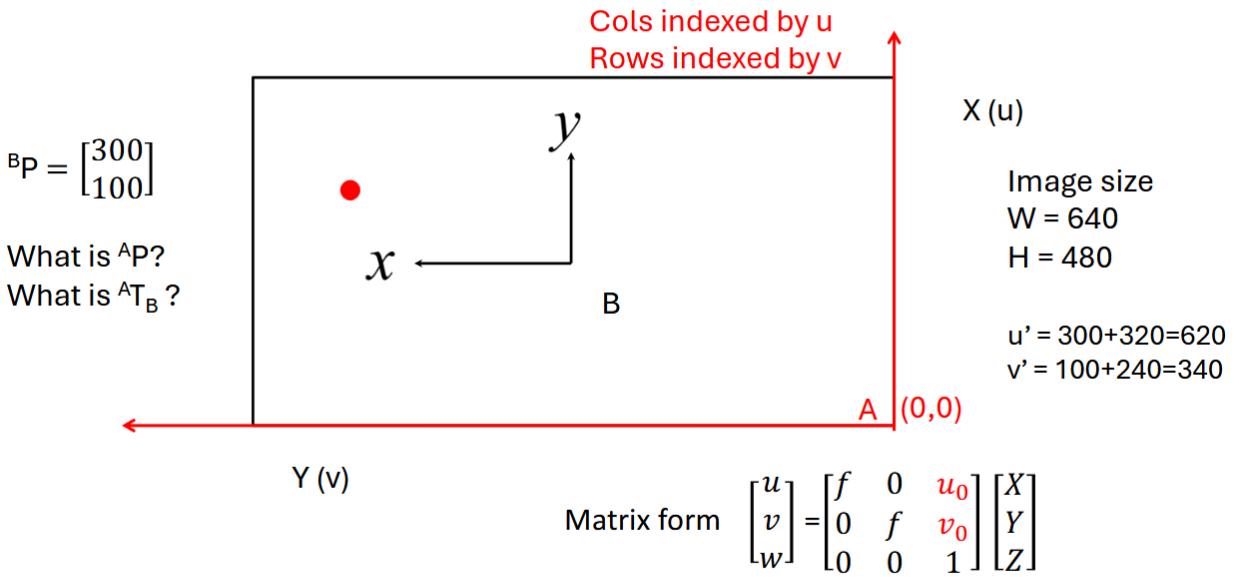
Image Coordinate

u = 300
v = 100



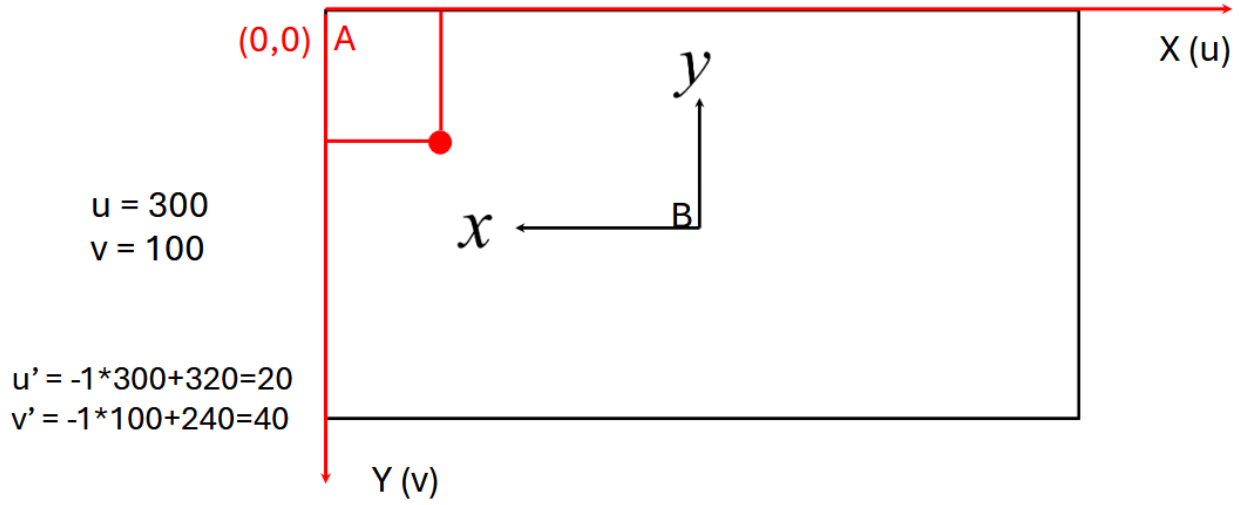
Until now, we use 2D coordinate conventions that are **consistent** with the 3D camera coordinate. However, if your application uses a different 2D coordiante, you'll need to further transform the (u, v).

For example, consider the following cases where we change the direction of the axes and the position of the origin.

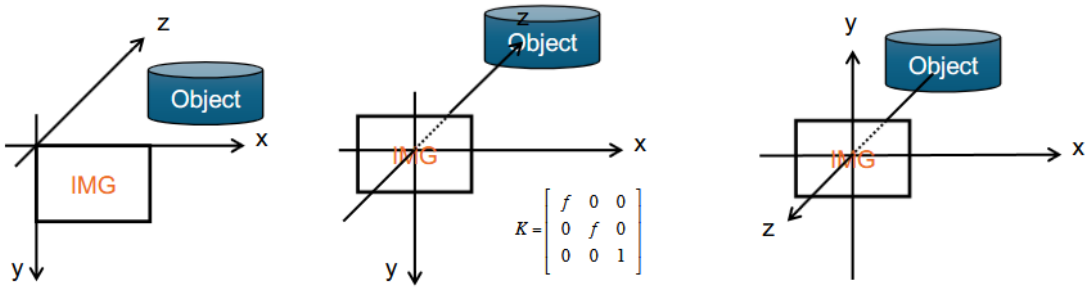


$${}^{B}P = \begin{bmatrix} 300 \\ 100 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



Popular Camera Coordinate Systems



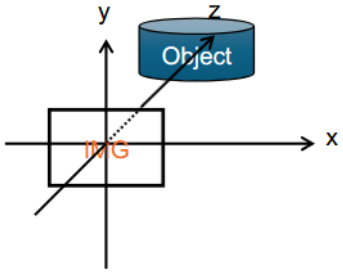
Computer Vision Researchers

Lazy Grad Students

Graphics Guru ☺ OpenGL

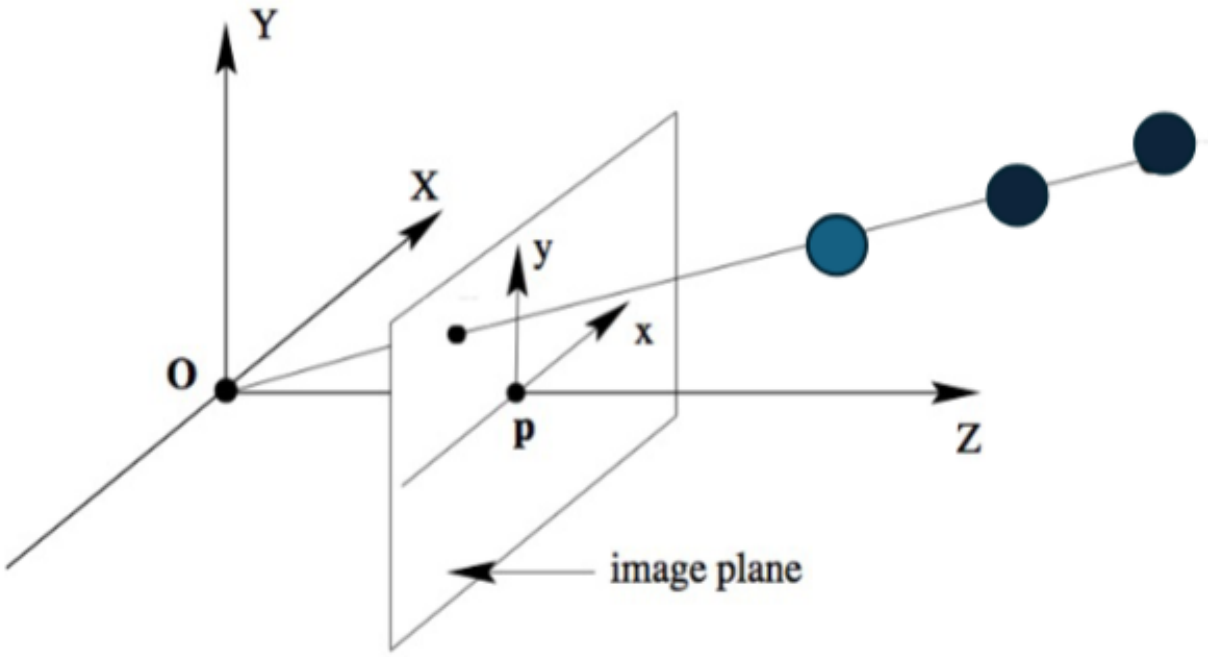
Right-Handed Coordinate System

Evil Microsoft DirectX ☺
= Left-Handed Coordinate System



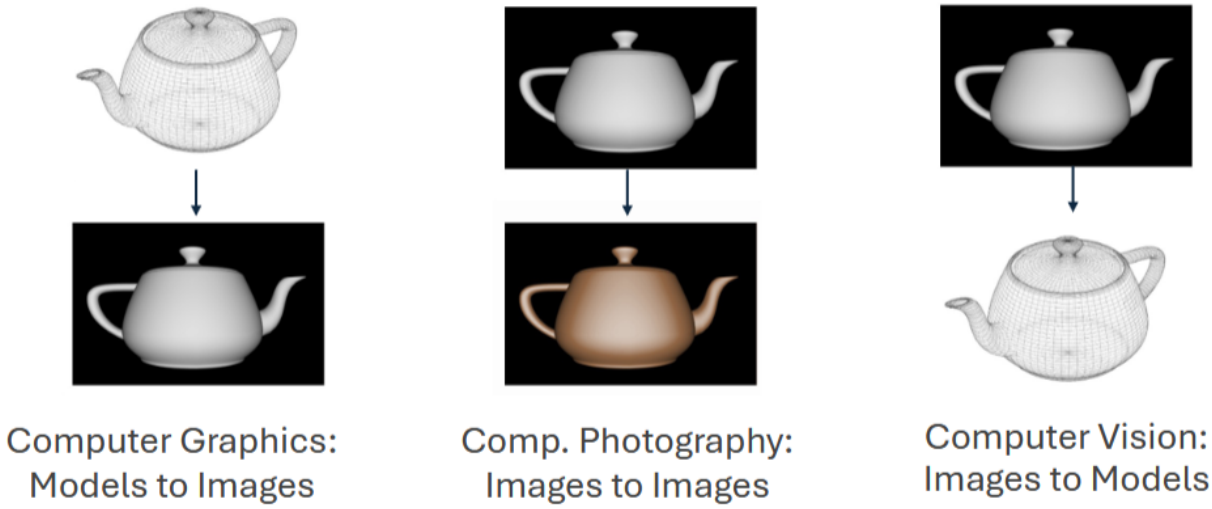
Camera Projection:

$$\lambda \begin{bmatrix} u \\ v \\ f \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



Computer Vision

A quick overview: what is computer vision?



What Makes 2D Computer Vision Hard?

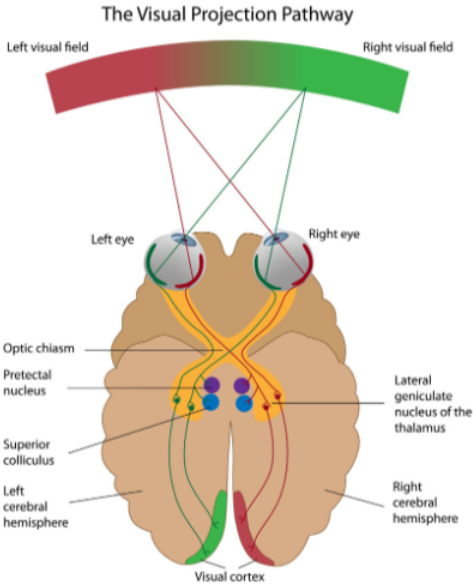
Variation: same cat, different poses, view points, ...

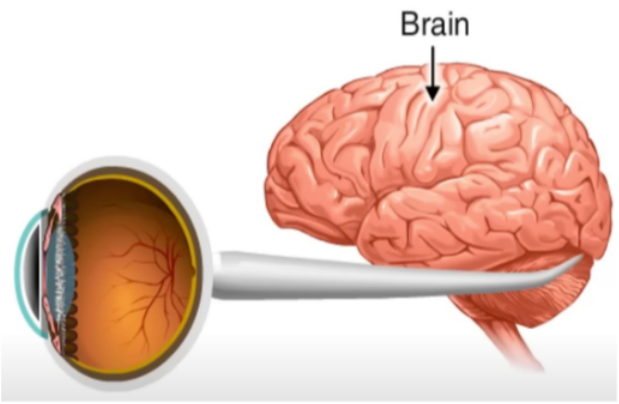
More variation: different cats, different shapes, colors, textures, ...

Other factors:

- Illumination
- Occlusion: partial observation
- Ambiguity: some objects may look a lot like others; different perspectives may look like different objects

Human Vision





3D Vision

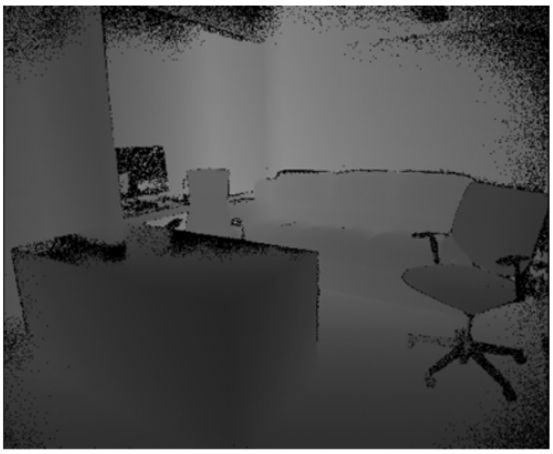
How do humans and animals perceive depth?

- Binocular vision: 2 eyes instead of 1
- Structure from Motion (SIM): walking around an object allows you to build a 3D model.

How do robots perceive depth?

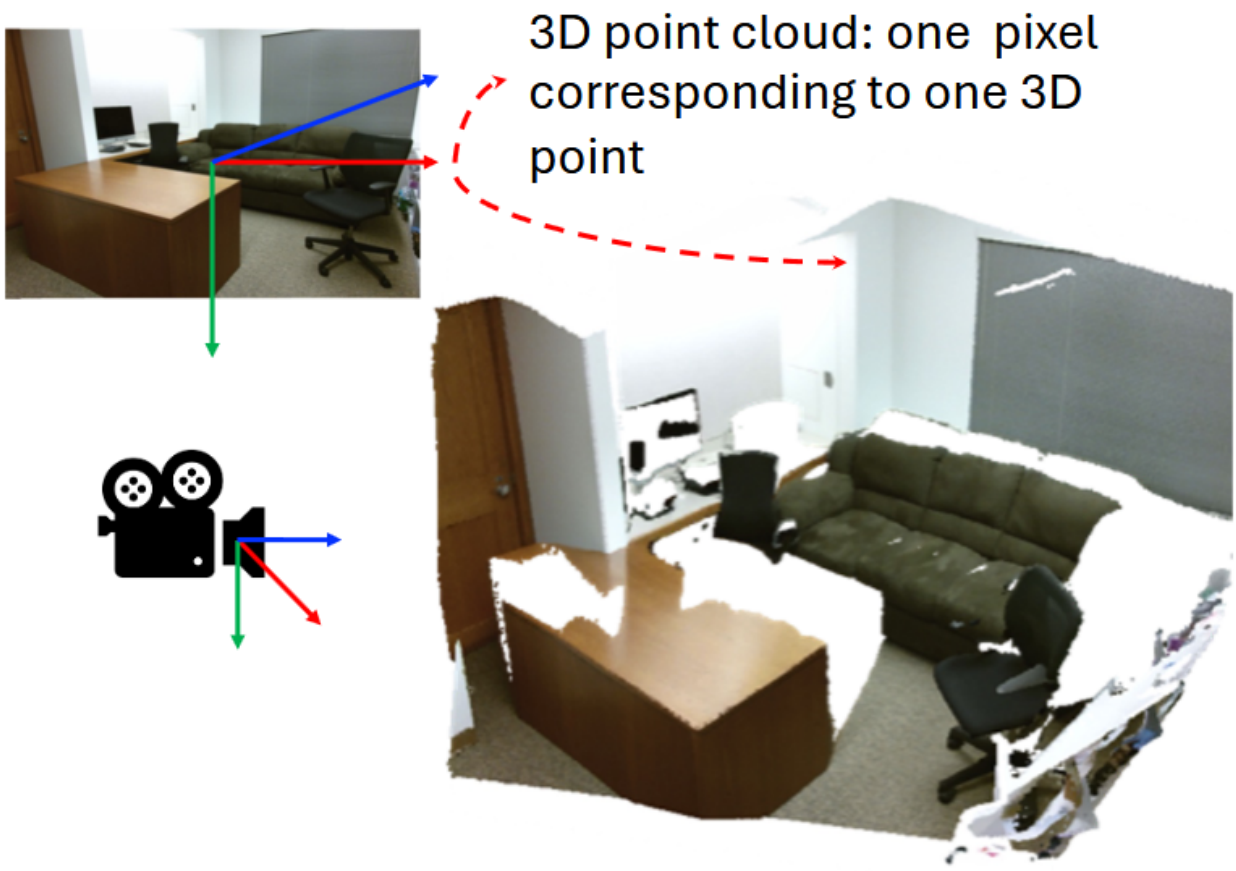
- Stereo camera
- Time of flight
- Structured light

2D to 3D Projection



- Knowing just 2D coordinate (u, v), we don't have enough information to compute the 3D point location (X, Y, Z)
- However, with an additional depth channel we can. (RGB-D image).
  - The image on the right is an RGB-D image. Each pixel records the depth value Z (in meter or millimeter)

We can combine the two images to form a single, 3D image.



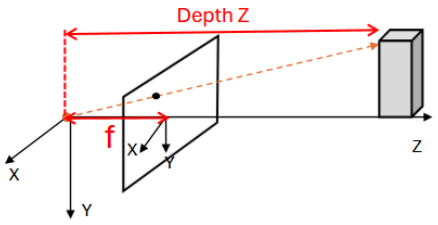
- Depth image → 3D point clouds:

- A pixel with
  - image coordinate (u, v)
  - Depth value $= Z$
  - Focal length f
- Its 3D location (X, Y, Z) in camera coordinate can be computed by:

$$X = \frac{u}{f} \cdot Z \qquad\qquad Y = \frac{v}{f} \cdot Z$$

- Z: reading from the depth image

Summary:



3D point (X,Y,Z) → 2D image coordinate (u,v)

$$u = \frac{fX}{Z} \qquad v = \frac{fY}{Z}$$

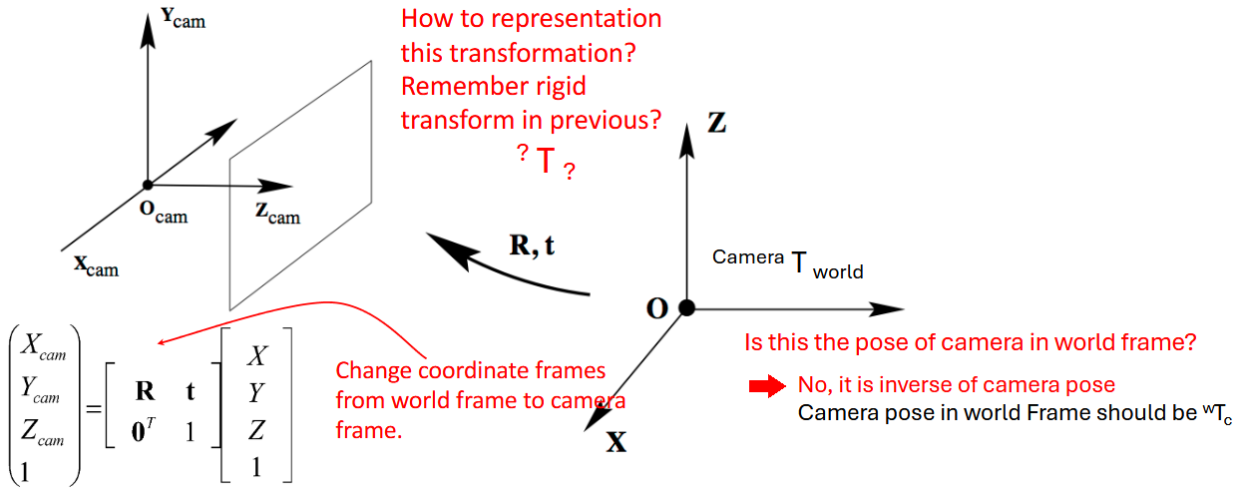
Depth Image (u,v, Z) → 3D Point Clouds

$$X = \frac{u}{f} * Z, \qquad Y = \frac{v}{f} * Z$$

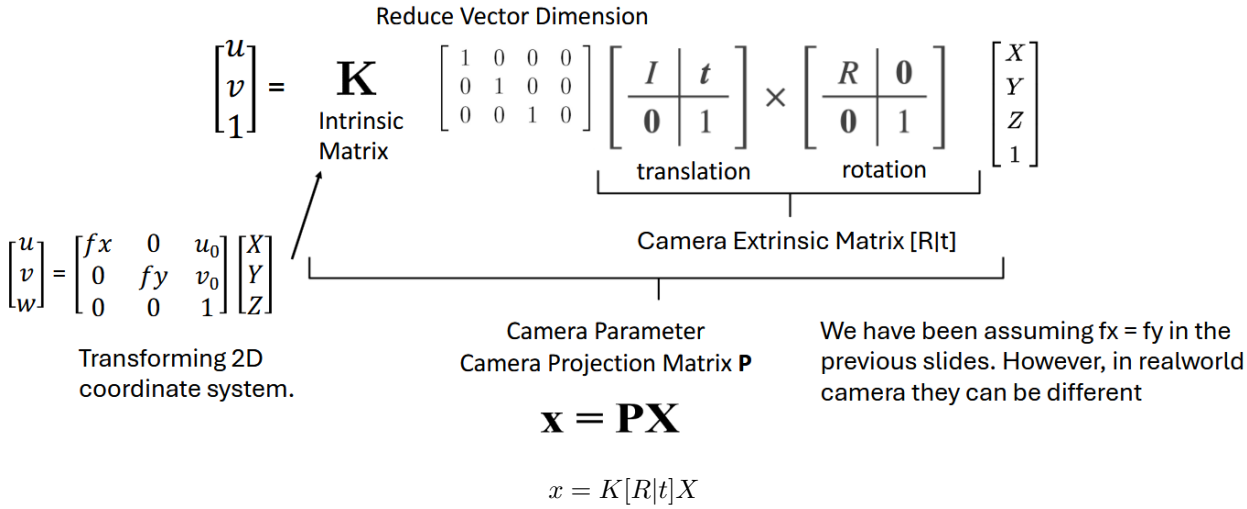
(Z : reading from depth image)

World Coordinate to Camera Coordinate

- In order to apply the camera model we described so far, the 3D point (X, Y, Z) must be expressed in camera coordinates (i.e.; centered at the camera origin)
- However, the world coordinate can be different from the *camera coordinates*.
- Requires an additional transformation



Camera: Putting Everything Together

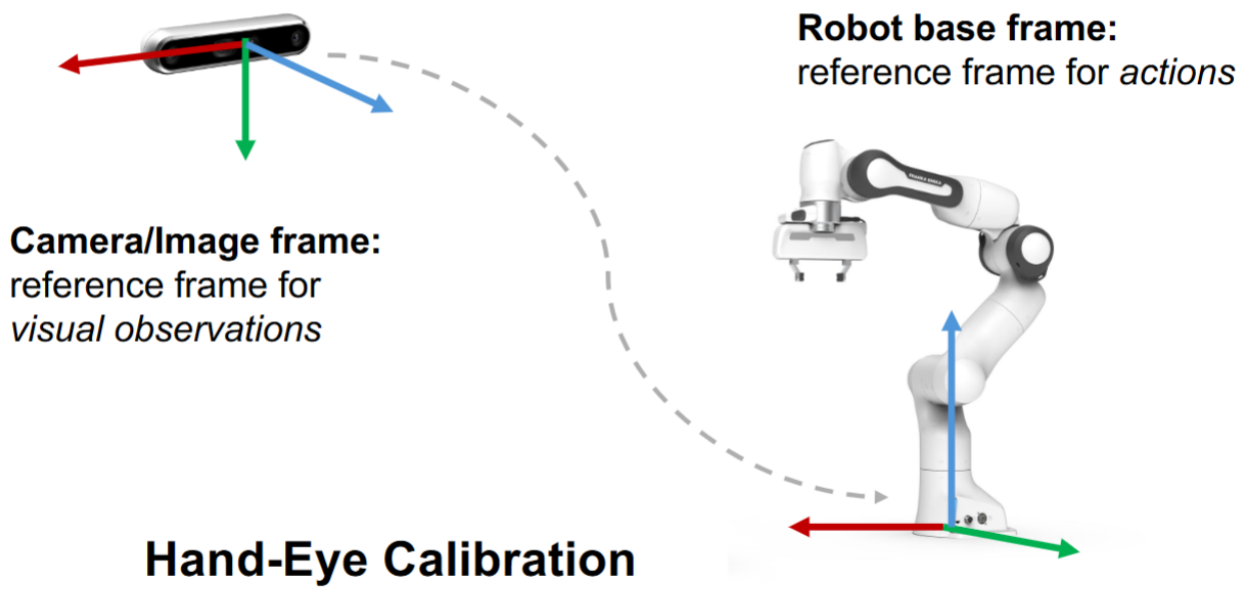


$$\mathbf{x} = \mathbf{PX}$$

$$x = K[R|t]X$$

- Map a 3D point X into a 2D coordinate in image x
- How to describe its *pose* in the world? (extrinsic matrix)
- How to describe its internal parameters? (intrinsic matrix)

Camera: Calibration



Goal: estimate the camera parameters

- Version 1: solve for projection matrix

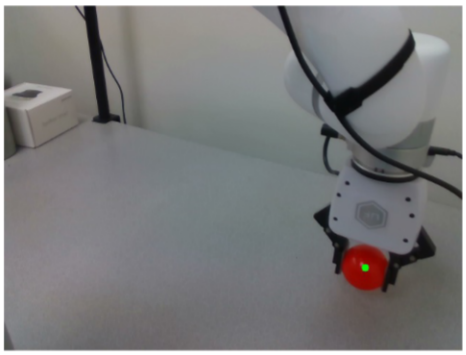
$$X = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = PX$$

- Version 2: solve for camera parameters separately
  - Intrinsic (focal length, principle point, pixel size)
  - Extrinsic (rotation angles, translation)

During the hand-eye calibration process, we move robot to a known position in its base frame, and captures the image with camera and detects the robot's hand's position (approximated with the red ball)

Robot: move_to(x=0.3, y=-0.2, z = 0.1)

Camera:



To calibrate:

1. Identify correspondance between image and scene

2. Compute mapping from scene to image

Requirement

1. Must know geometry very accurately
2. Must know correspondance

$$x_i = PX_i$$

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

You want to find a **rotation matrix** $\mathbb{R} \in SO(3)$, and a **translation vector** $t \in \mathbb{R}^3$ such that $p_i^B \equiv Rp_i^A + t$. (This is the matrix above.)

The Kabsch-Umeyama Algorithm

1. Compute centroids
$$\bar{p}^A = \frac{1}{N}\sum_{i=1}^{N} p_i^A, \qquad \bar{p}^B = \frac{1}{N}\sum_{i=1}^{N} p_i^B$$
2. Center the points
$$q_i^A = p_i^A - \bar{p}^A, \qquad q_i^B = p_i^B - \bar{p}^B$$
3. Compute the cross-covariance matrix
$$H = \sum_{i=1}^{N} q_i^A (q_i^B)^T$$
4. Apply SVD (singular value decomposition)
$$H = U\Sigma V^T$$
5. Construct the rotation matrix
$$R = VU^T$$
   - If $\det(R) < 0$, correct for reflection:
   - $V[:, 3] \leftarrow -V[:, 3]$, (flip the third column of V)
6. Compute the translation vector
$$t = \bar{p}^B - R\bar{p}^A$$
7. Output the transformation (combine rotation and translation into a rigid body transformation)

Pseudo Code

```
// compute centroids
centroid_A = mean of A_points
centroid_B = mean of B_points

// center point sets
A_centered = [Ai - centroid_A for Ai in A_points]
B_centered = [Bi - centroid_B for Bi in B_points]

// Compute cross-covariance matrix H:
```

```
H = zero 3×3 matrix
for i = 1 to N:
    H += outer_product(A_centered[i], B_centered[i])

// Perform Singular Value Decomposition:
[U, Σ, V_T] = SVD(H)

// Compute rotation matrix R:
R = V_T^T * U^T

// Handle reflection case (det(R) < 0)
if determinant(R) < 0:
    V_T[2] = -V_T[2] # flip the 3rd row of V_T
    R = V_T^T * U^T

// Compute translation vector t:
t = centroid_B - R * centroid_A

RETURN R, t
```

Computer Vision Basics

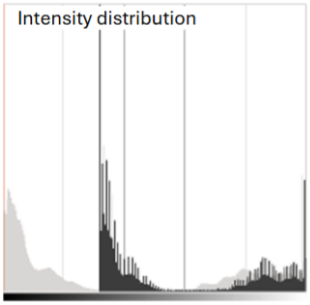
Basic Image Processing

There are many operations we can apply on a image. Here are some examples:

- Resize
- Color manipulation
- Contrast
- Brightness
- Gamma
- Filtering
- Denosing

Brightness (overall intensity of an image)

Adjust brightness: $g(x) = f(x) + \beta$
Increasing (or decreasing) the β value will add (or subtract) a constant value to every pixel. Pixel values outside of the [0, 1] range will be saturated (i.e., a pixel value higher or lower than 1 or 0 will be clamped to 1 or 0).



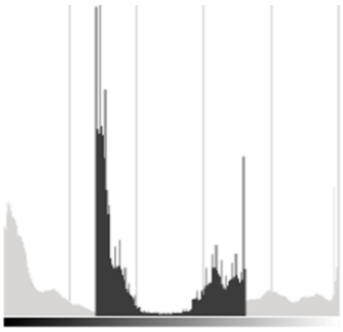
In light gray, histogram of the original image, in dark gray when brightness = 0.15

f(x): input image pixel value

g(x): output image pixel value

Contrast

Adjust contrast: $g(x) = \alpha f(x)$
The α parameter will modify how the intensity distribution spread. If $\alpha < 1$, the color levels will be compressed and the result will be an image with less contrast.



In light gray, histogram of the original image, in darker gray when contrast a<1

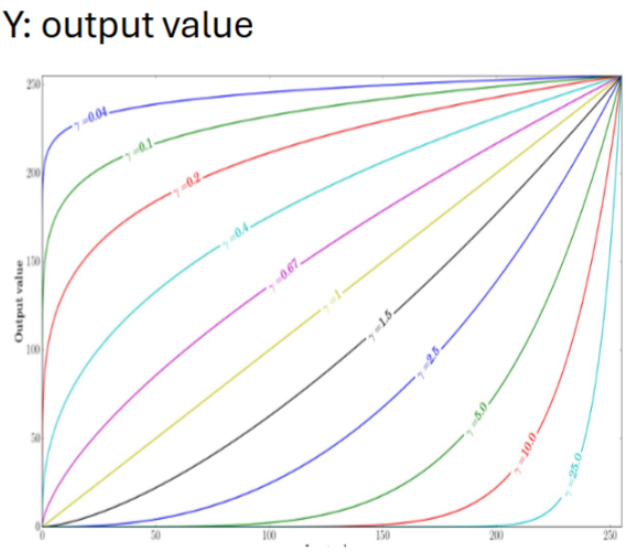
f(x): input image pixel value

g(x): output image pixel value

Note: f(x) is normalize to have zero mean, g(x) will be convert back to (0,1) before visualization

Gamma Correction

- Gamma correction can be used to correct the brightness of an image by using a nonlinear transformation between the input values and the mapped output values: $g(x) = [f(x)]^{\frac{1}{\gamma}}$
- As this relation is nonlinear, the effect will not be the same for all the pixels and will depend to their original value.
- When $\gamma < 1$, the original dark regions will be brighter and the histogram will be shifted to the right. (reducing contrast)



The following image has been corrected with: α=1.3 and β=40.
The overall brightness has been improved but the clouds are now greatly saturated due to clipping.

The following image has been corrected with: gamma correction γ=0.4.

Image Filtering

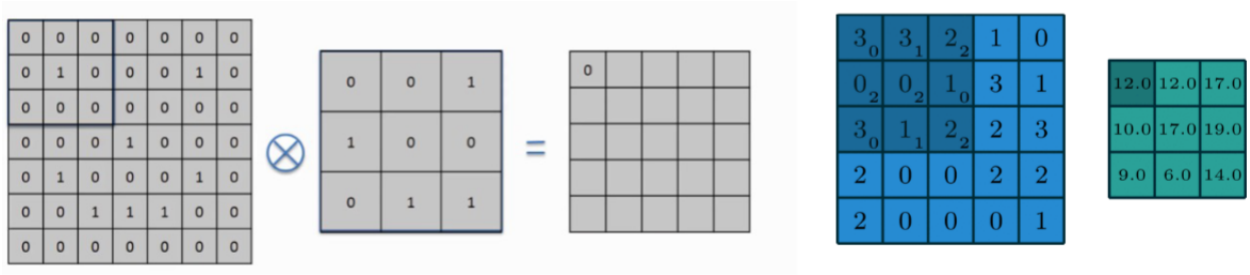
Image filtering: compute function of local neighborhood at each position

- Filtering can be used for:
  - Enhance images (denoise, increase contrast, etc.)
  - Extract information from images (textures, edges, distinctive points, etc.)
  - Detect patterns (template matching)
  - Deep convolutional networks

Convolution Operation

- Sliding the kernel (small 2D matrix) over the 2D image (big 2D matrix)
- Performing elementwise multiplication with the part of the input it is currently on

- Sum up the result into a single output pixel
- Slide to the next location



Padding

- What about near the edge?
  - the filter window falls off the edge of the image
- Boundary padding
  - Zero (smaller value around boundary)
  - Circular/warp (best for panorama)
  - Replicate
  - Symmetric

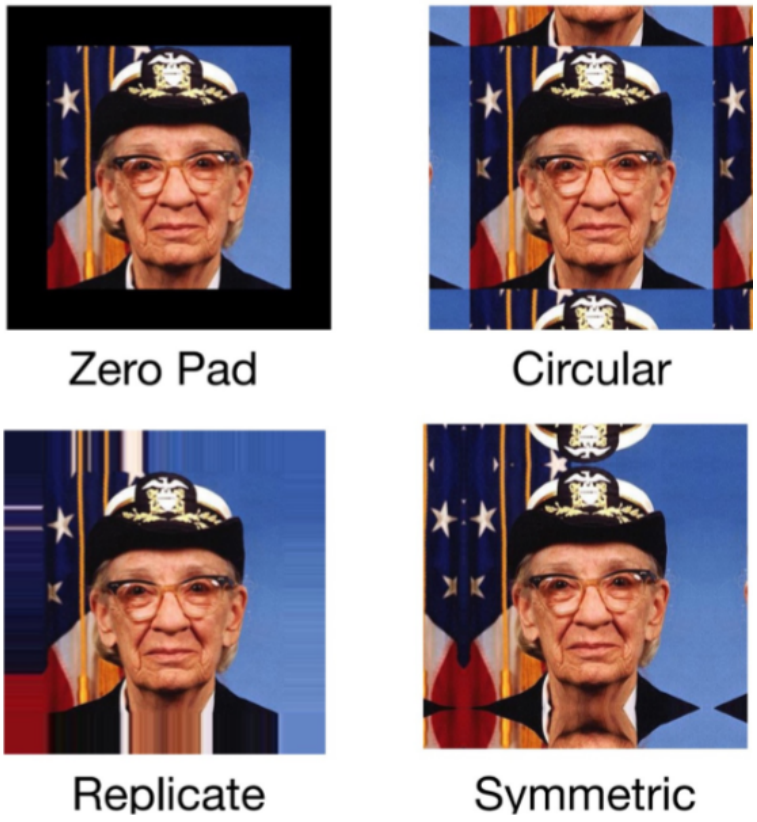
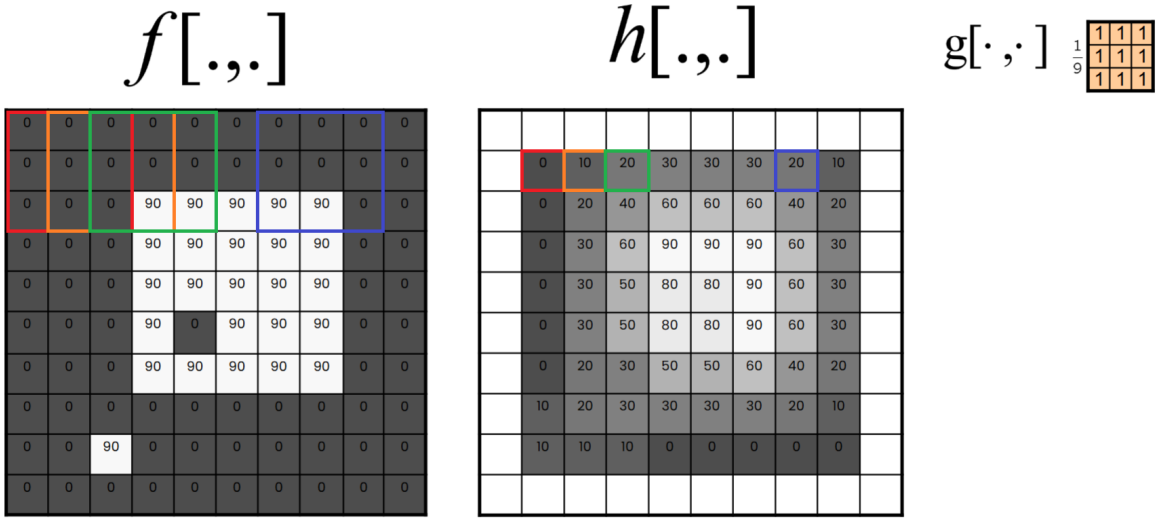


Image Filtering



Dot product at each position

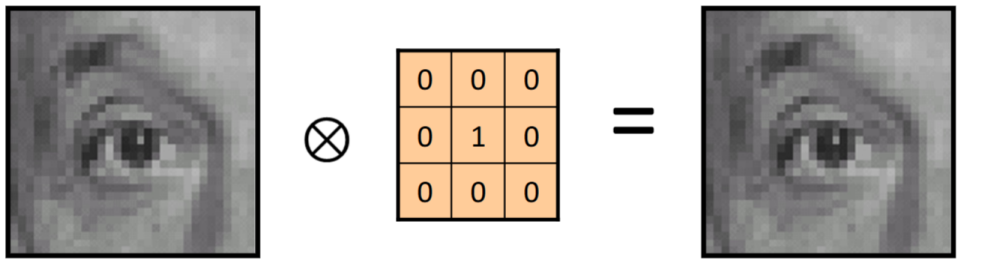
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering: Box Filter

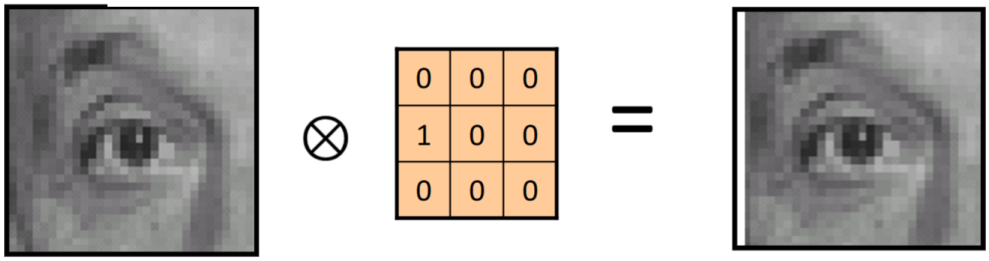
- What does it do?
  - Replaces each pixel with an average of its neighborhood
  - Achieve smoothing effect (remove sharp features)

Example: Convolution Kernels



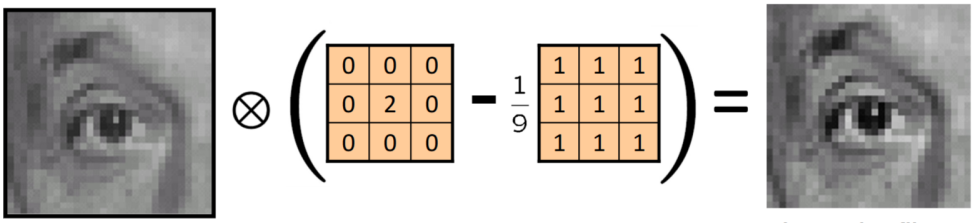
Original

Identical image



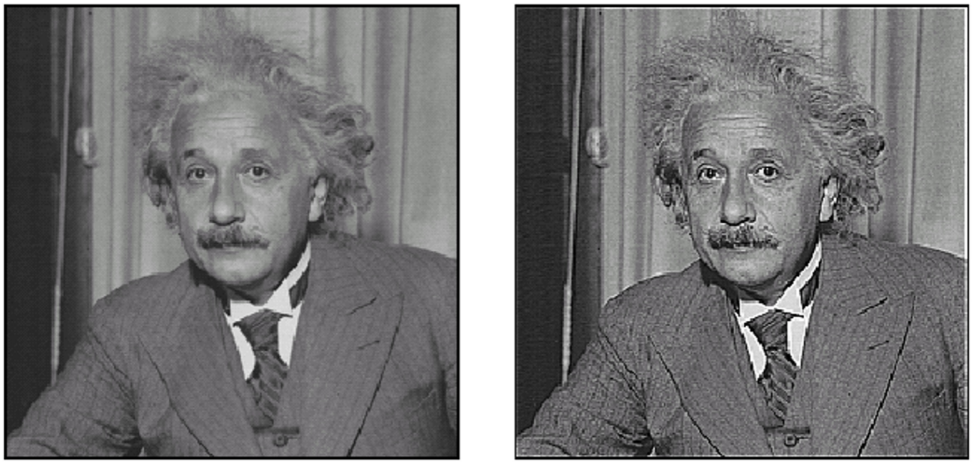
Original

Shifted right
By 1 pixel



Original

Sharpening filter
(Amplifies the difference)

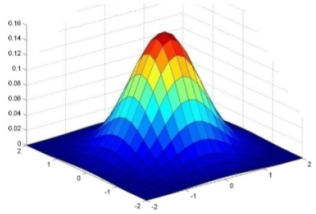


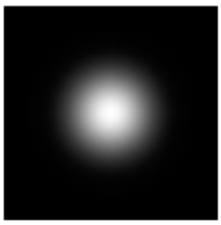
before **after**

Source: D. Lowe

Other Filters

Gaussian Kernel





Approximated by:

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

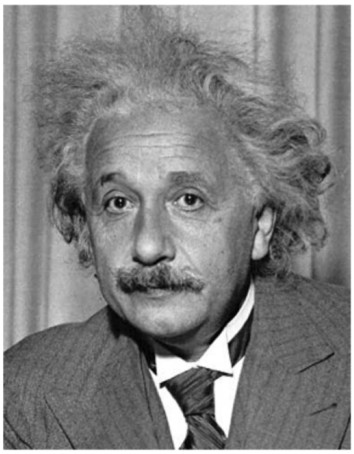
5 x 5, $\sigma = 1$

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Constant factor at front makes matrix sum to 1

Source: C. Rasmussen

Edge Detection



Vertical Edge (absolute value)

Horizontal Edge (absolute value)

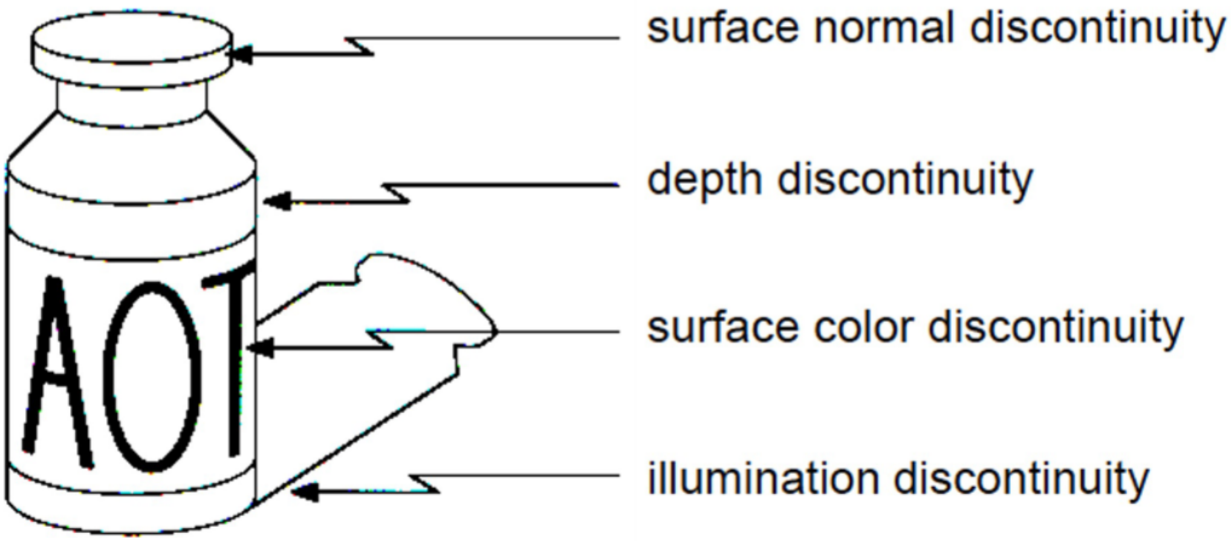
1	0	-1
2	0	-2
1	0	-1

Vertical Sobel

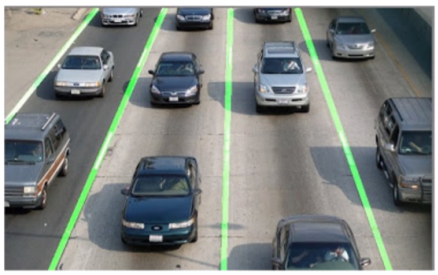
1	2	1
0	0	0
-1	-2	-1

Horizontal Sobel

- Goal: Identify sudden changes (discontinuities) in an image
- Why? Intuitively, edges carry a lot of the semantic and shape information from the image.

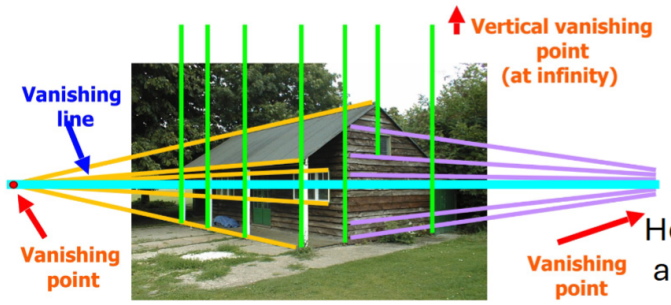


Edge Detection in Robotics



Extract information
Lane detection

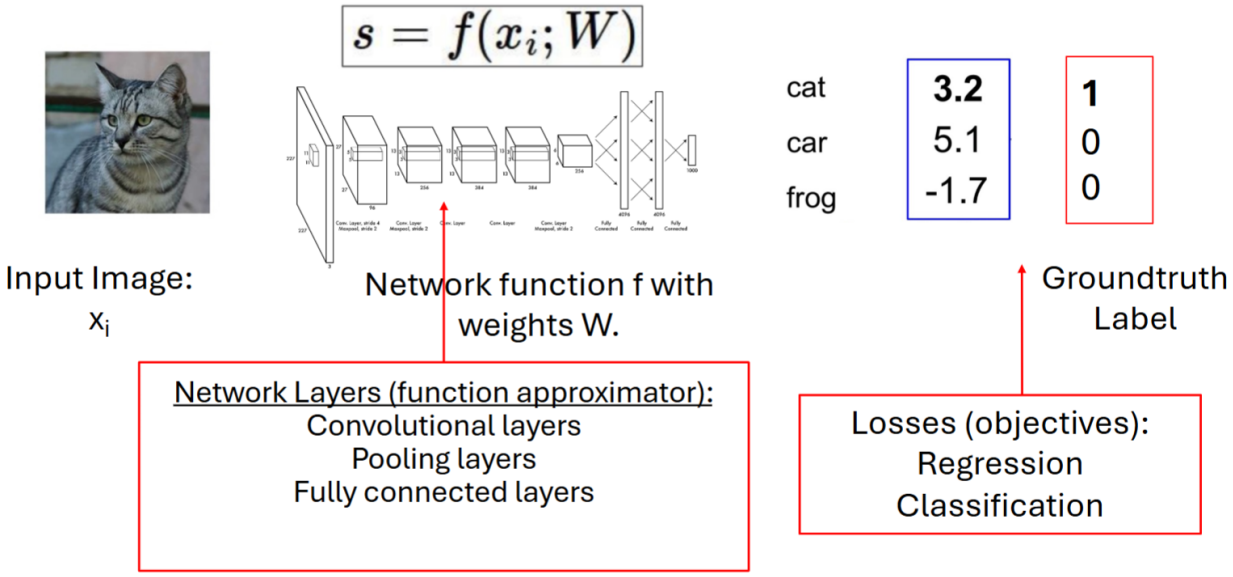
Can we automatically learn which filter to apply to get most useful information?
→
Goal of **convolutional neural network (CNN)**



Help recover geometry and camera viewpoint

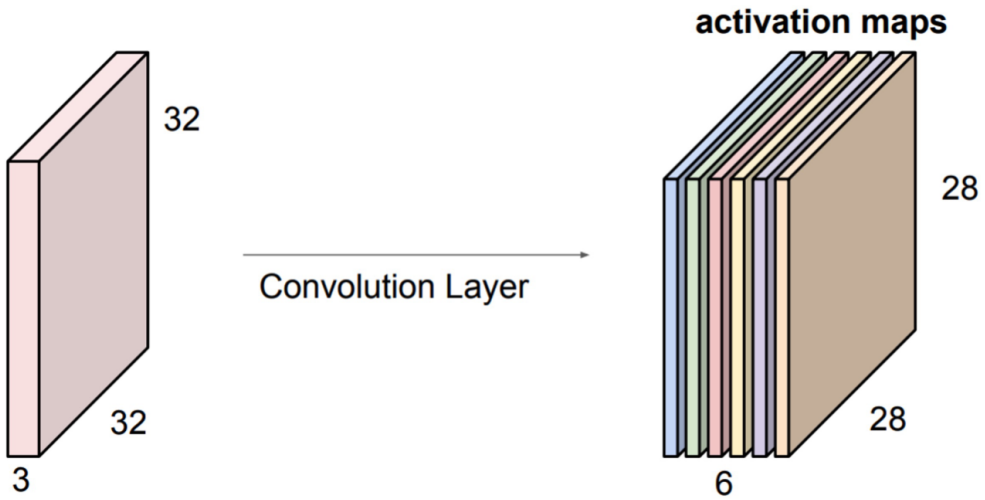
Convolution Neural Networks

Convnet Building Blocks



Convolutional Layer

- Input: an image
- Processing: convolution with **multiple** filters
- Output: an image, (# channels = # filters), weighted sum of input
- `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, padding_mode='zeros')`
- **Convolve** the filter with the image i.e., "slide over the image spatially, computing dot products"
- A convolutional layer can have multiple filters.



If we had **6** 5x5 filters, we'll get 6 separate activation maps. We stack these up to get a "new image" of size 28x28x6!

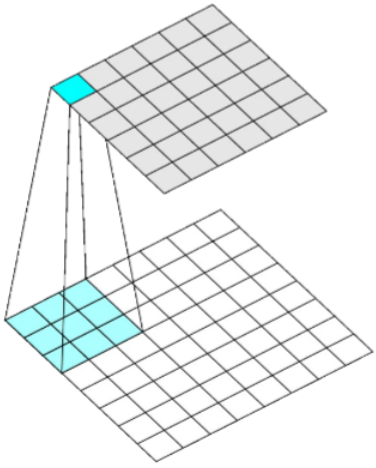
Kernel Size

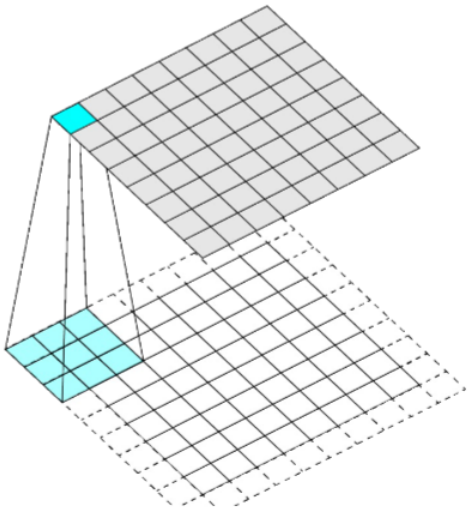
How big the filter for a layer is typically between 1x1 and 7x7.

- Larger kernel aggregate information from larger local region (bigger receptive field)
- 1x1 is just linear combination of channels in previous image (no spatial processing)
- Filters usually have the same number of channels as the input image. (e.g., an RGB image has 3 channels, so one filter is a $n \times n \times 3$ rather than just a 2d kernel)

Padding

- Convolutions have problems on edges
- Do nothing: output a little smaller than input
- Pad: add extra pixels on edge





Stride

- How far to move filter between applications
- We've done stride 1 convolutions up until now, approximately preserves image size
- Could move filter further, downsample image

Input - Output Size

- Input $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$ image
- `Conv2d(K, N, S, P)`: Number of kernels `K`, kernel size $[$`N` $\times$ `N`$]$, stride `S`, padding `P`
- What's the output feature size (C_{out}, H_{out}, W_{out})?
  - $H_{\text{out}} = \lfloor \frac{H_{\text{in}} + 2 \cdot \text{padding} - (N-1) - 1}{\text{stride}} + 1 \rfloor$
  - $W_{\text{out}} = \lfloor \frac{W_{\text{in}} + 2 \cdot \text{padding} - (N-1) - 1}{\text{stride}} + 1 \rfloor$
  - $C_{\text{out}} = K$

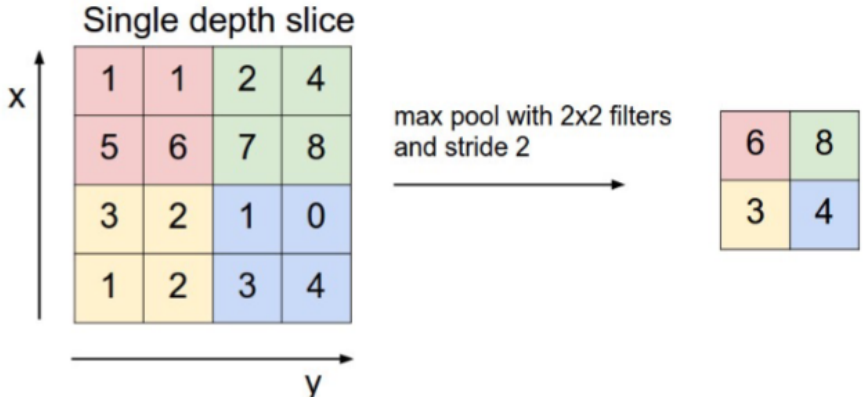
Pooling Layer

- Input: an image
- Processing: pool pixel values over region
- Output: an image, shrunk by a factor of the stride

Hyperparameters:

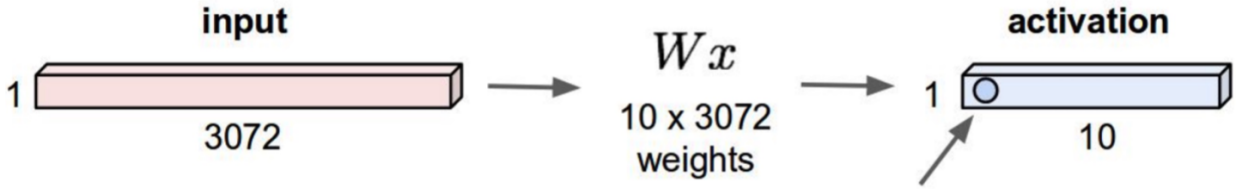
- What kind of pooling? Average, mean, max, min
- How big of stride? Controls downsampling
- How big of region? Usually not bigger than stride

Most common: maxpooling 2x2 stride of 2, or 3x3 stride of 3.



Fully Connected Layer (InnerProduct)

Every input neuron connects to every output neuron



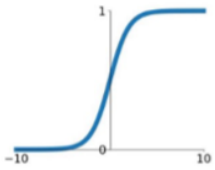
- Often used to go from image feature map → final output or map image features to a single vector
- Eliminates spatial information

Activation Layer

- Used to increase non-linearity of the network without affecting receptive fields of the conv layers.
- Most commonly used activation layer: ReLU

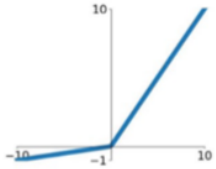
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



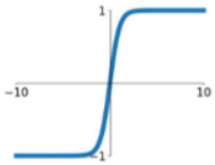
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

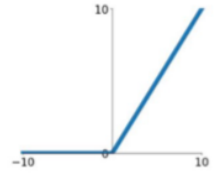


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

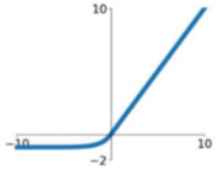
ReLU

$$\max(0, x)$$



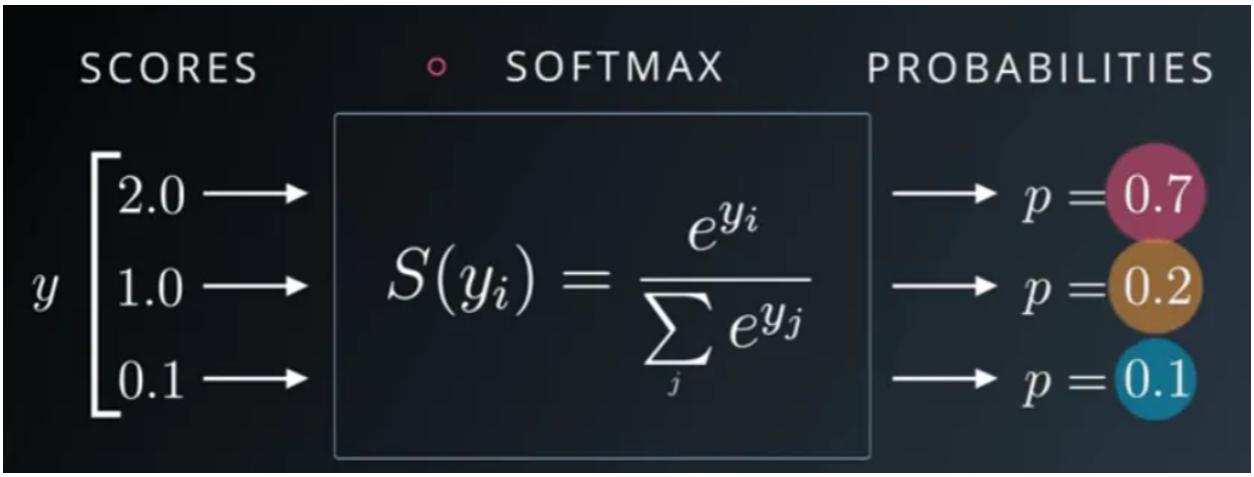
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Layer: Softmax

- A special kind of activation layer, usually at the end of fully connected layer and before cross-entropy loss.
- Can be viewed as a fancy normalization function that produce a probability distribution vector



Loss Function

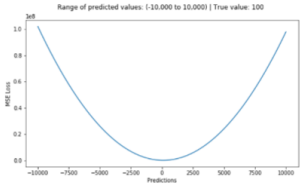
A loss function scores how far off a prediction is from the desired "target" output.

- loss(prediction, target) returns a number called "the loss"
- Big Loss = Bad Error
- Small Loss = Minor Error
- Zero Loss = No Error

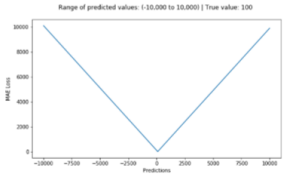
Regression	Classification
Predict real-valued output	Predict category output
What temperature will it be tomorrow?	Will it be sunny tomorrow? Often predicts continuous values which are probabilities of different labels, use the one with max probabilities as predicted class.
Loss function: squared error, L1 error, ...	Loss function: cross-entropy

Regression

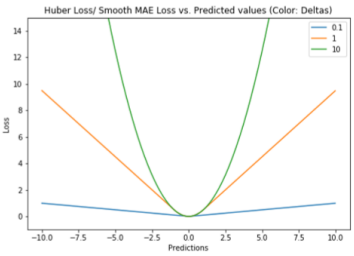
L2 Loss $MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$



L1 Loss $MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$



L2 when $\delta \sim 0$ and L1 when $\delta \sim \infty$ (large numbers.)

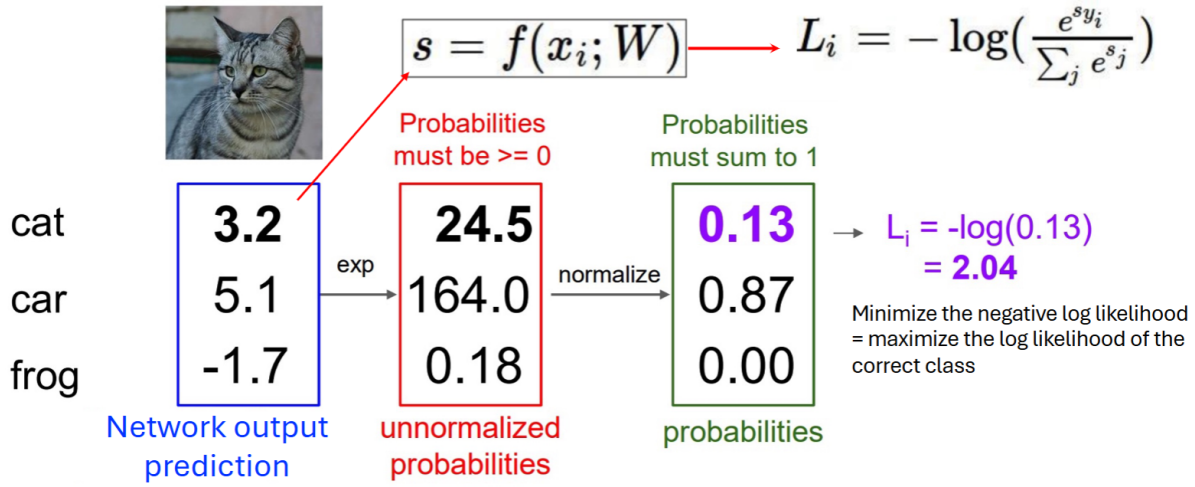


Smooth L1 $L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$

L1 loss is more robust to outliers, but its **derivatives around zero are not continuous**, making it inefficient to find the solution. L2 loss is **sensitive to outliers**, but gives a more stable and closed form solution.

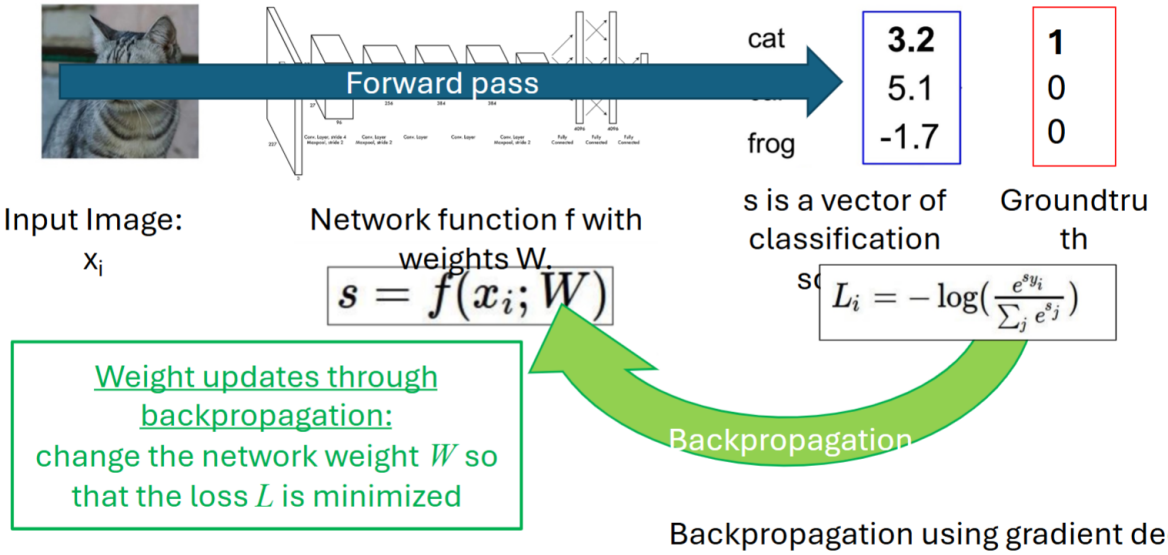
- Smooth L1 is somewhat a combination of the two.

Classification: CrossEntropy



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

- The network output prediction is compared to the ground truth vector



Backpropagation using gradient decent:
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

- Finally, weights update through backpropagation.
- Change the network weight W so that the loss L is minimized.

Summary of Convnet Building Blocks

- Convolutional layers:
  - Used convolution operation to extract features
  - Convolution with stride > 1 will downsample image
- Pooling layers:

  - Used to downsample feature maps, making processing more efficient
- Fully connected layers:
  - Often used as last layer, to map image features → prediction
  - No spatial information
  - Inefficient: lots of weights, no weight sharing
- Activation layers: increase non-linearity of the network
- Loss function: classification and regression