Image Analysis for Engineering Design Applications

EE 493, Fall 2022-2023

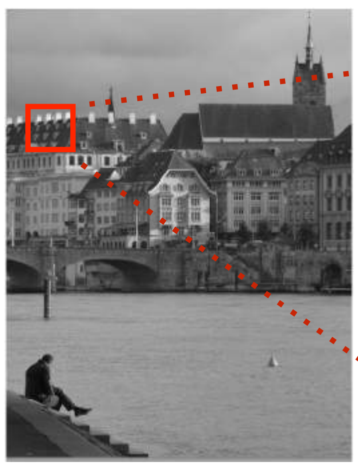
Middle East Technical University
Department of Electrical and Electronics Engineering

Outline

- Digital image representations
- Object and pattern detection
- 3D geometry and perspective correction

Image Representations

- A digital image is an array of numbers.



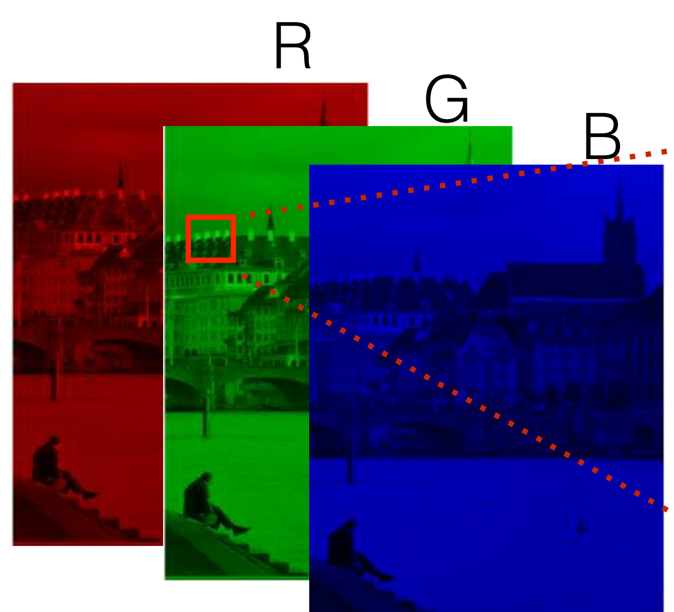
Grayscale image

	12	13	14	15	16
214	49	51	53	50	49
215	45	46	47	45	44
216	82	81	77	71	67
217	111	111	109	105	96
218	86	84	85	79	69
219	108	105	94	66	43
220	123	117	97	63	47
221	118	104	74	51	48
222	109	81	56	51	48
223	88	60	49	49	47
224	65	51	49	48	47
225	50	46	47	46	46
226	47	45	46	47	46

With 8 bits, pixel values change between 0 and 255

Color image

=



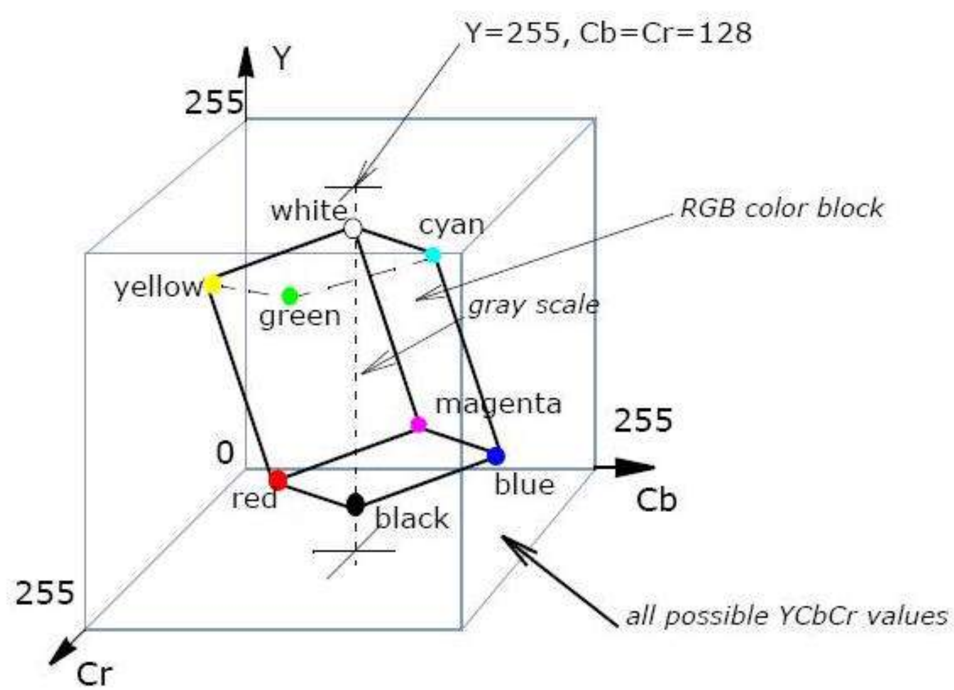
76	77	76	77	83	81
73	79	81	83	83	80
68	69	72	76	74	75
72	70	76	76	71	73
72	75	79	71	71	70
74	76	74	73	77	71
74	76	72	70	75	68
70	73	71	69	75	75
68	73	75	71	68	68
66	65	67	68	69	73
67	67	69	72	73	76
62	66	71	76	78	77

Color Spaces

Alternative color spaces can be preferred in different applications:

- YCbCr color space: Describe images in terms of luminance and chrominance components

$$
\begin{array}{r l}
\text{Luminance} & \{ \\
\text{Chrominance} & \{
\end{array}
\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}
$$



[Intel integrated performance primitives for intel architecture reference manual]

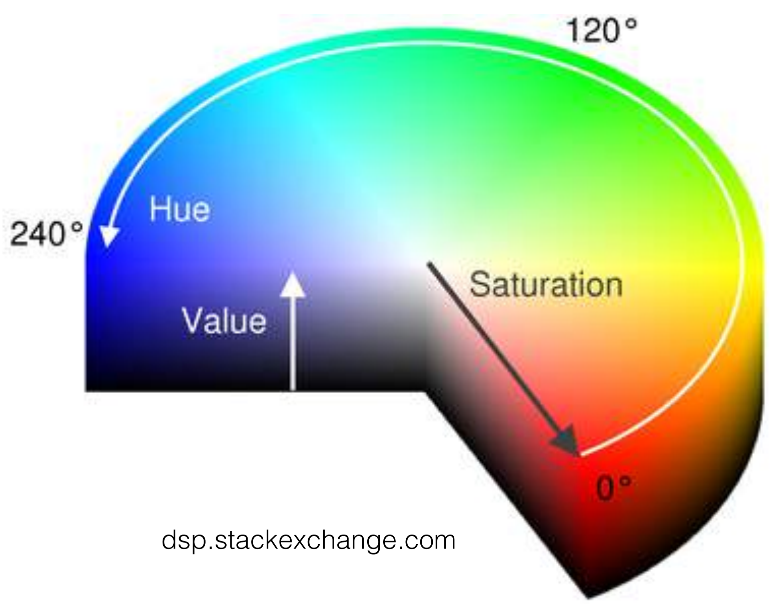
- Often used in image and video compression applications

Color Spaces

- HSV color space: Describe images in terms of hue, saturation, and value

$$H = \begin{cases} 0^\circ & \text{, if } \max = \min \\ 60^\circ \times \frac{G-B}{\max - \min}, & \text{, if } \max = R \\ 60^\circ \times \left(\frac{B-R}{\max - \min} + 2\right), & \text{if } \max = G \\ 60^\circ \times \left(\frac{R-G}{\max - \min} + 4\right), & \text{if } \max = B \end{cases}$$

$$S = \begin{cases} 0 & \text{, if } \max = 0 \\ \frac{\max - \min}{\max} & \text{, if } \max \neq 0 \end{cases} \qquad V = \frac{\max}{255}$$



- Separates color information from intensity
  - ➡ More robust to illumination changes than RGB in color-based detection

Outline

- Digital image representations
- Object and pattern detection
- 3D geometry and perspective correction

Object Detection

- Object detection: Search for what characterizes your query object in the image

- Search this in your image:

[pixabay.com]

  - Color-based detection: “Look for a red object”

  - Shape-based detection: “Look for a sphere”

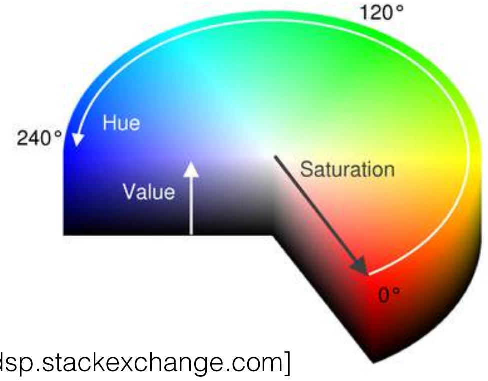
  - Color & Shape -based detection: “Look for a red sphere”

Color-Based Detection

- Color-based detection characterizes the query object in terms of its color.
- A threshold is applied.
- Example: To look for a red object

[pixabay.com]

  - R>200
  - R>200 & G<100 & B<100
  - $-60^\circ < \text{Hue} < 60^\circ$



[dsp.stackexchange.com]

Limitations of Color-Based Detection

- Color-based detection is easy to implement.
- However, it has **serious limitations**!

- Uncontrolled background is easily confused with the object


[dreamstime.com]

- Appropriate value of the threshold depends a lot on the illumination conditions

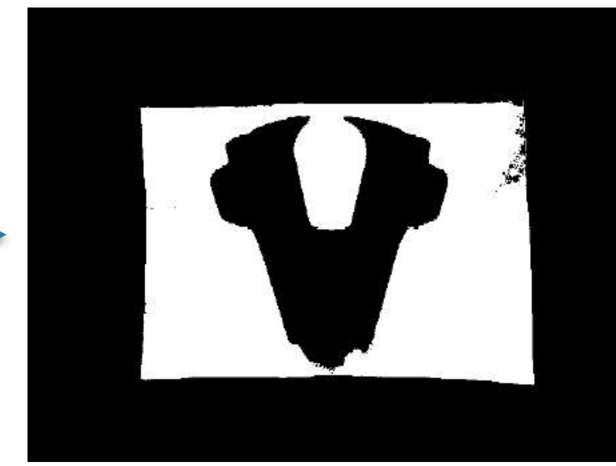
- Detection algorithm is quite vulnerable to illumination change, noise, shadows, …

Color-Based Detection Example

- Problem: Detect the red frame surrounding the gadget

Under daylight illumination

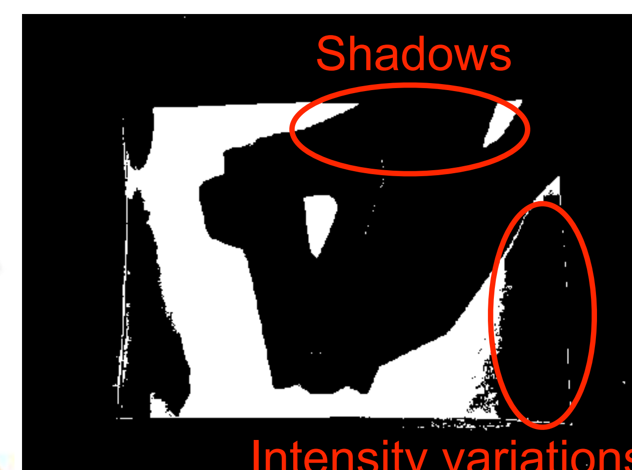
Detection with
R>200, B<180, G<180



Detection result

Under direct sunlight

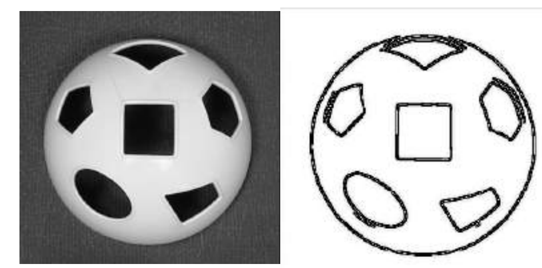
Detection with
R>200, B<180, G<180



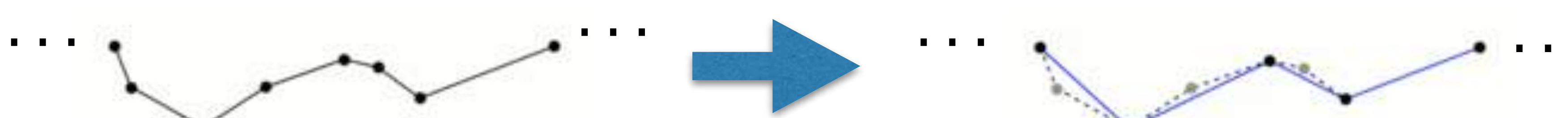
Detection result

Shape-Based Detection

- Problem: Look for a generic shape (rectangle, circle) in your image
- A basic contour-based shape detection algorithm:
  - Find the contours in the image



  - Simplify the contours by reducing points



[wikipedia.org]

  - Determine shape based on contour information:
    - 3 vertices: Triangle
    - 4 vertices: Rectangle
    - Many vertices + extra conditions: Circle

Shape-Based Detection

- Open-CV implementation of the contour-based shape detection algorithm is available:

[pyimagesearch.com]

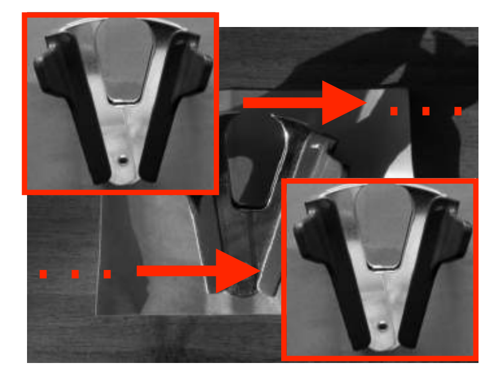
An Overview of Object Detection Techniques

- Color-based detection: Not robust to imaging conditions
- Shape-based detection: More reliable if you look for a simple shape
- Techniques for objects with more complex shapes:
    - Template matching
    - Feature matching
    - Customized detectors
    - Deep networks

Object Detection with Template Matching

- Template matching:
  - Convolve (correlate) the query pattern with the searched image

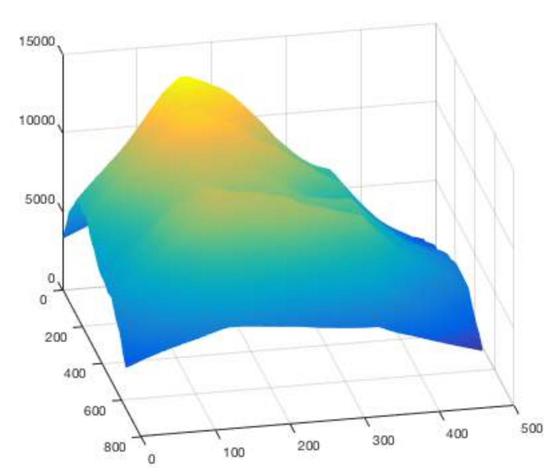
*

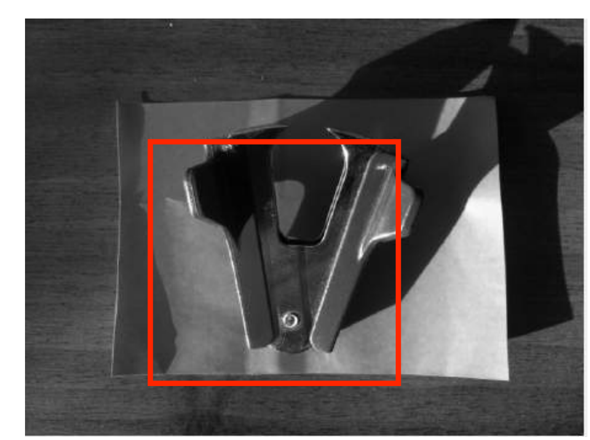


Query pattern

Searched image

  - Inspect the maximum value of the convolution

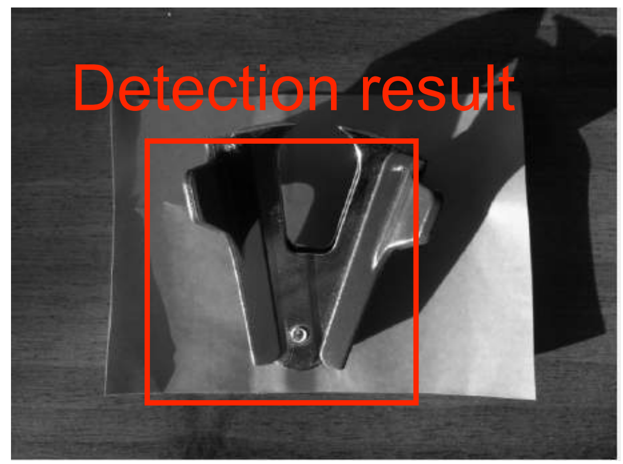




Object Detection with Template Matching

Query pattern: 

Searched image: 



- Advantages:
  - More reliable than color-based detection
  - Easy to implement (FFT-based implementations, Phase Correlation method)
- Limitations:
  - Geometric transformations (scale changes, rotations) lead to errors

Feature-Based Object Detection

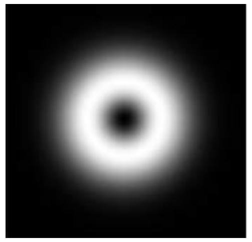
- Features: Points of interest in an image that can be repeatedly detected

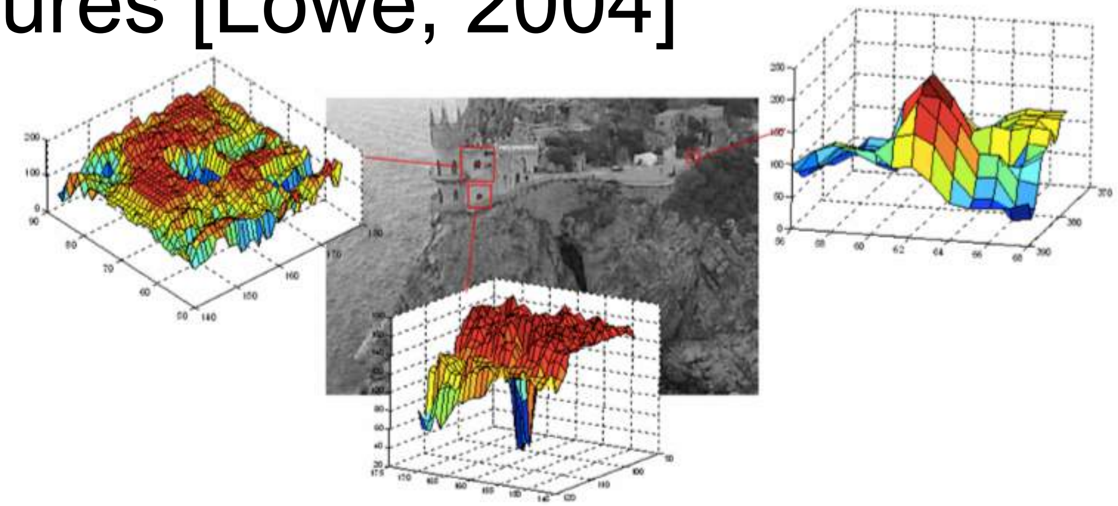
- Corner points, blob-like regions, ring-shaped regions …

Feature Detection Algorithms

- Harris corner detector: Looks for corner-like points where image gradients are high in both directions [Harris, 1988]

- SIFT (Scale-Invariant Feature Transform): Looks for ring-like structures [Lowe, 2004]

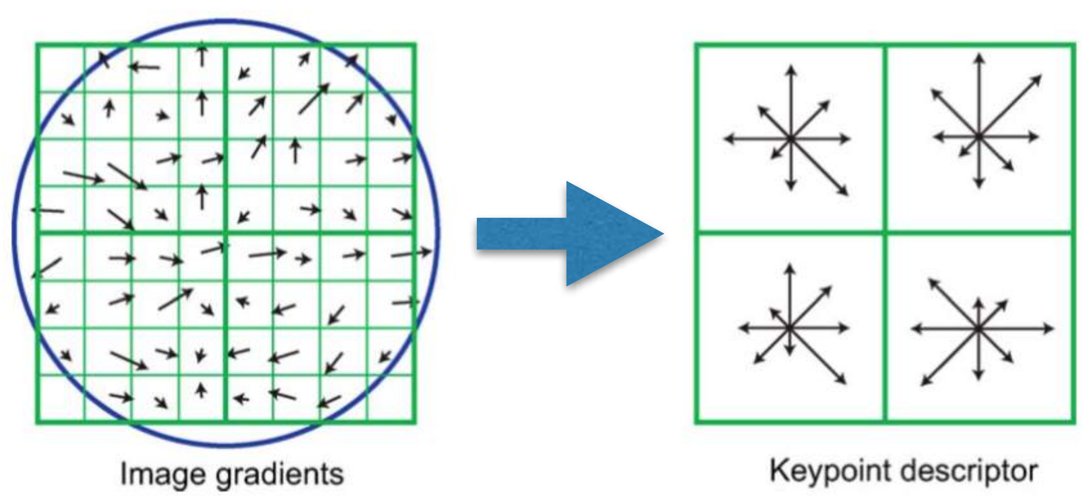




- Several improvements in more recent feature detectors: Faster operation, transformation-invariance,…
  - SURF, FAST, ORB, …

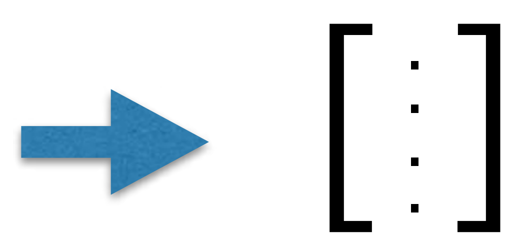
Feature Descriptors

- A descriptor vector is assigned to each feature point.
  - Describes the structure of the image around the feature
- Example: A common descriptor is Histogram-of-Gradients



[Lowe, 2004]

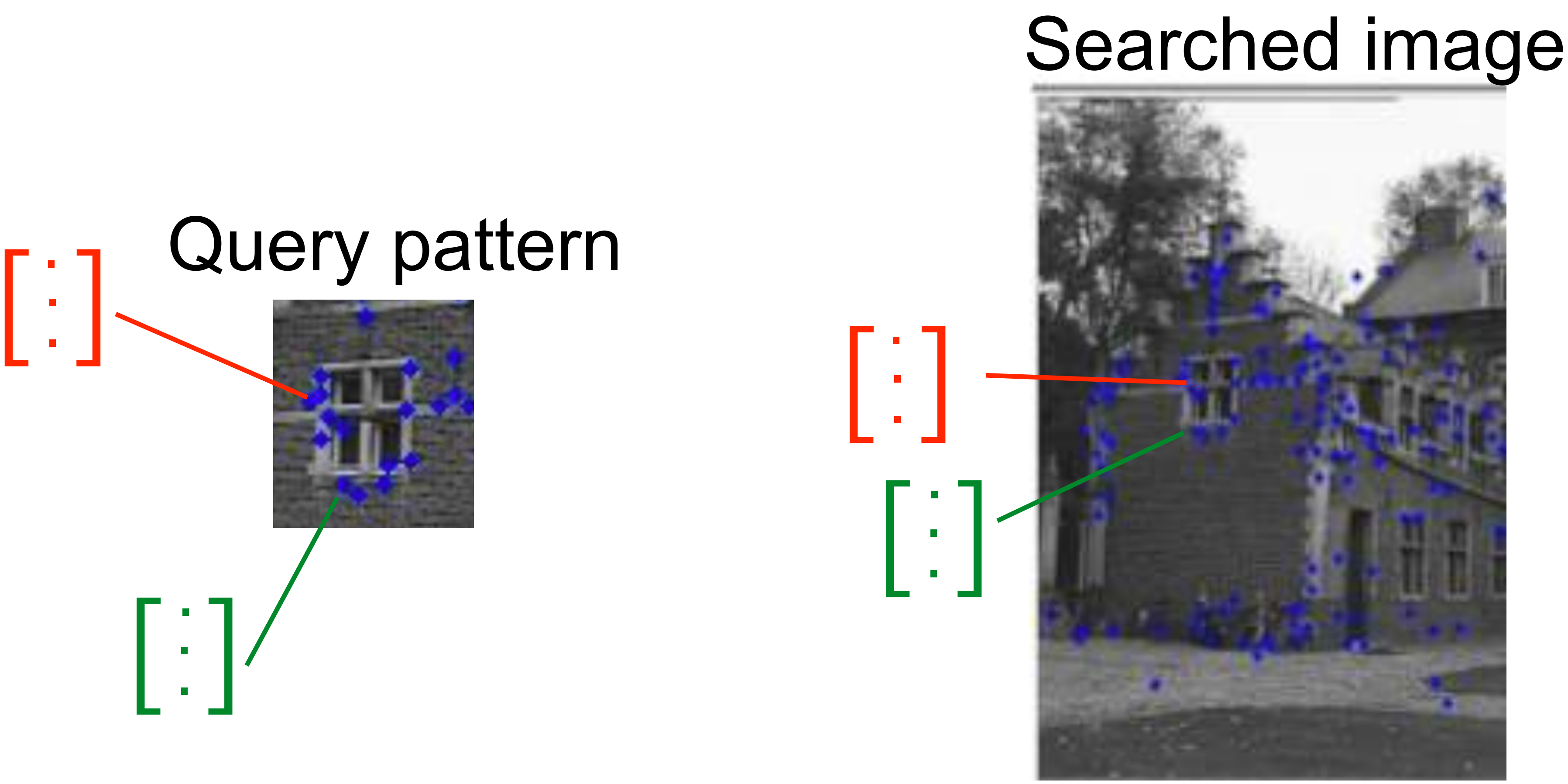
Histograms of
gradient directions



Combined histograms:
Descriptor vector

Feature Matching

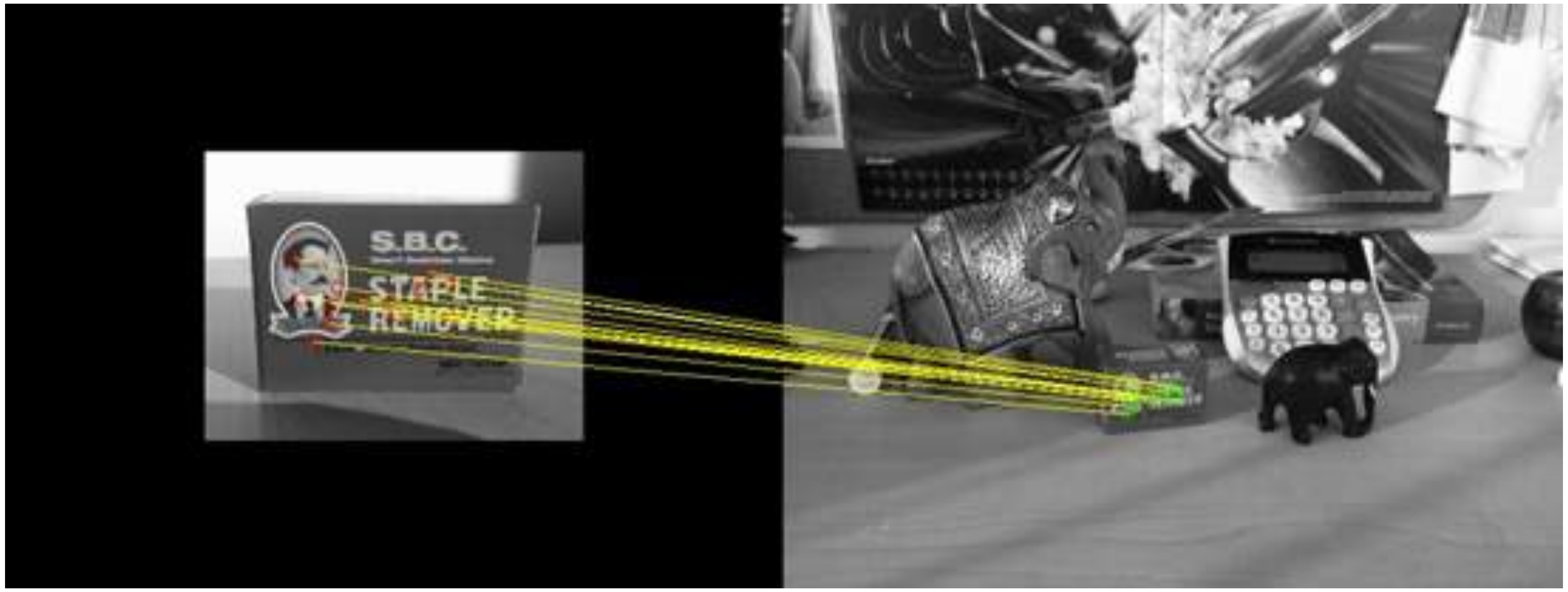
- Each feature is assigned a descriptor vector



- Comparing the descriptor vectors, the match of each query feature is found in the searched image.

Feature Matching

- Wrong matches are eliminated with algorithms like RANSAC [Fischler 1981].
- From the matched features, the query pattern is detected in the searched image.
- The location of the sought object can be estimated.

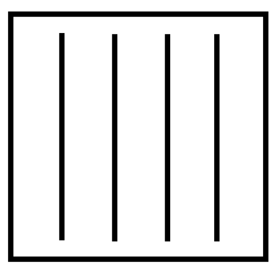


[www.mathworks.com]

- Open-source implementations are available

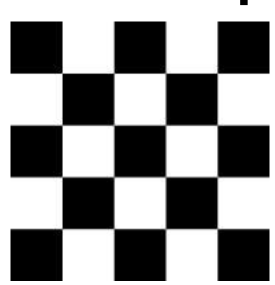
Customized Detectors

- It may be possible to devise your own detector based on the properties of the object you look for.
  - Searched pattern:



Develop an object detector based on edge detection

  - Searched pattern:

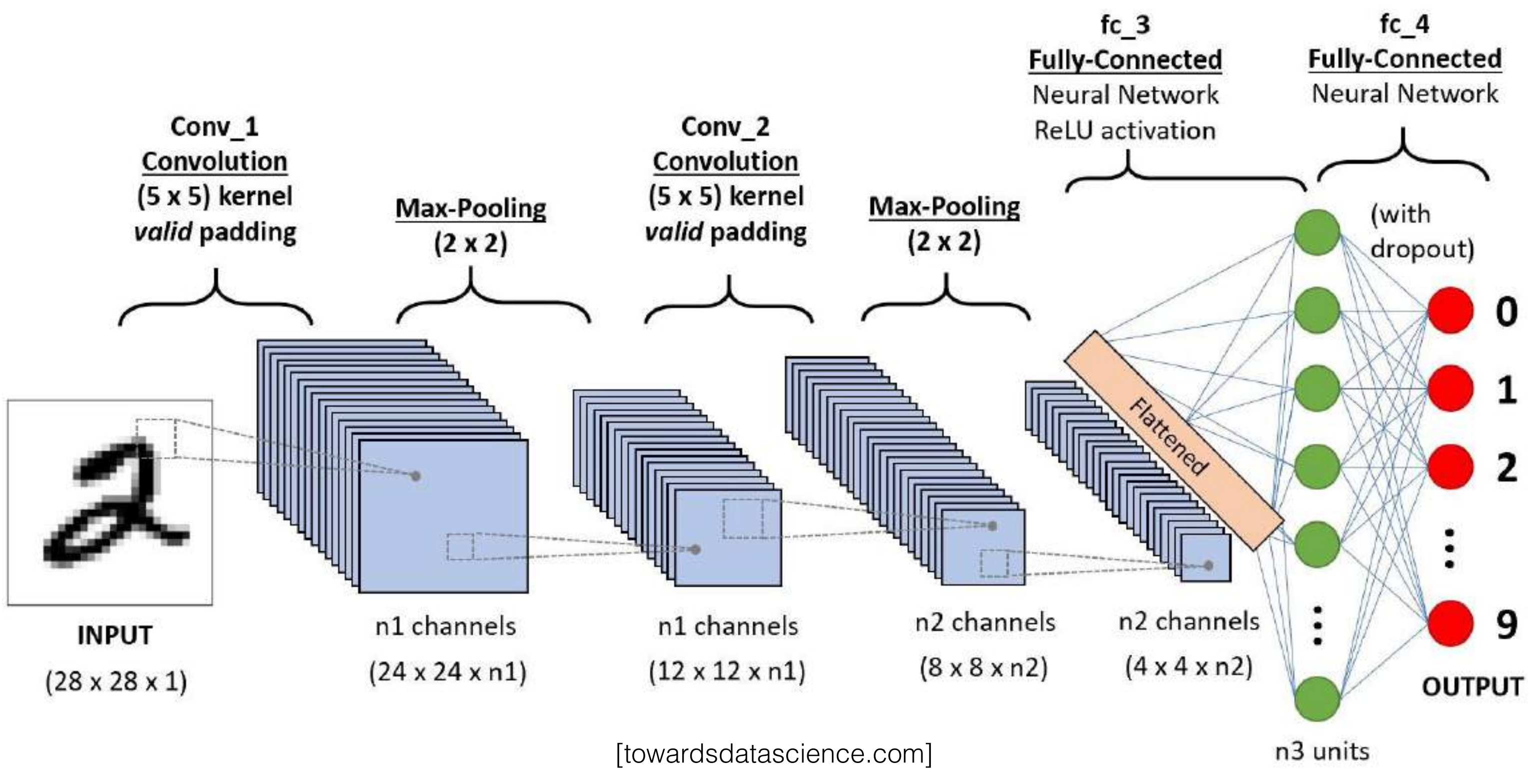


Develop an object detector based on corner detection

- Examples: QR code scanners, barcode scanners, …
- Bonus: The pattern may allow the inference of extra information (orientation, distance, etc.)

Deep learning methods

- Object detection based on deep networks: Active research topic
- Classical structure of a Convolutional Neural Network (CNN):



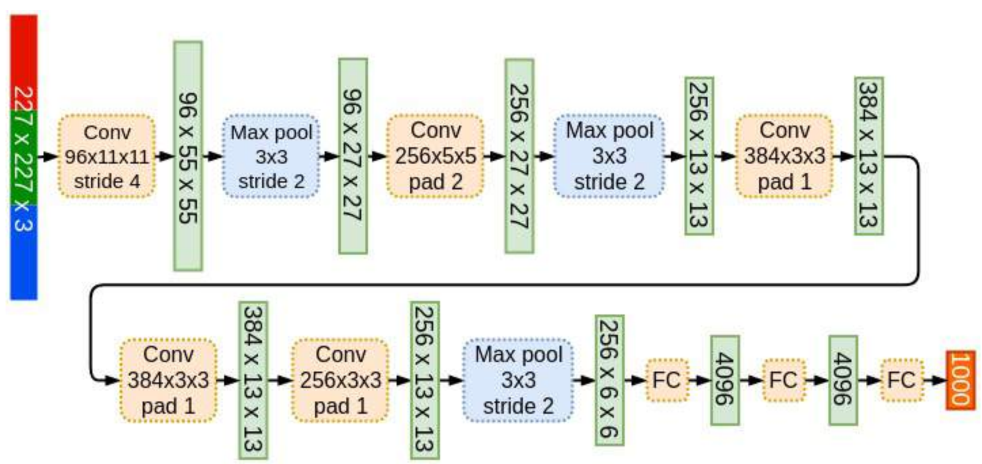
[towardsdatascience.com]

- Training the network = Learning the filter coefficients in each layer
- **Number of parameters** to learn is proportional to:

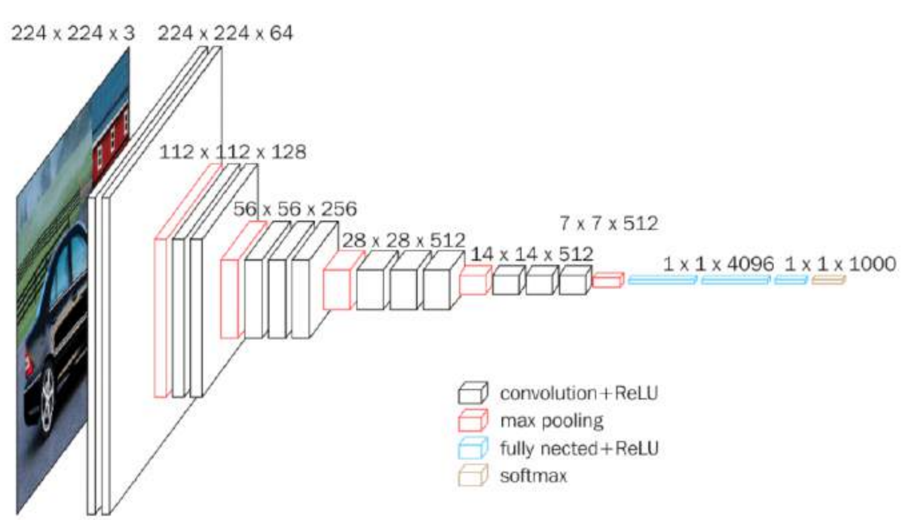
  Filter size * Number of channels * Number of layers

Deep learning methods

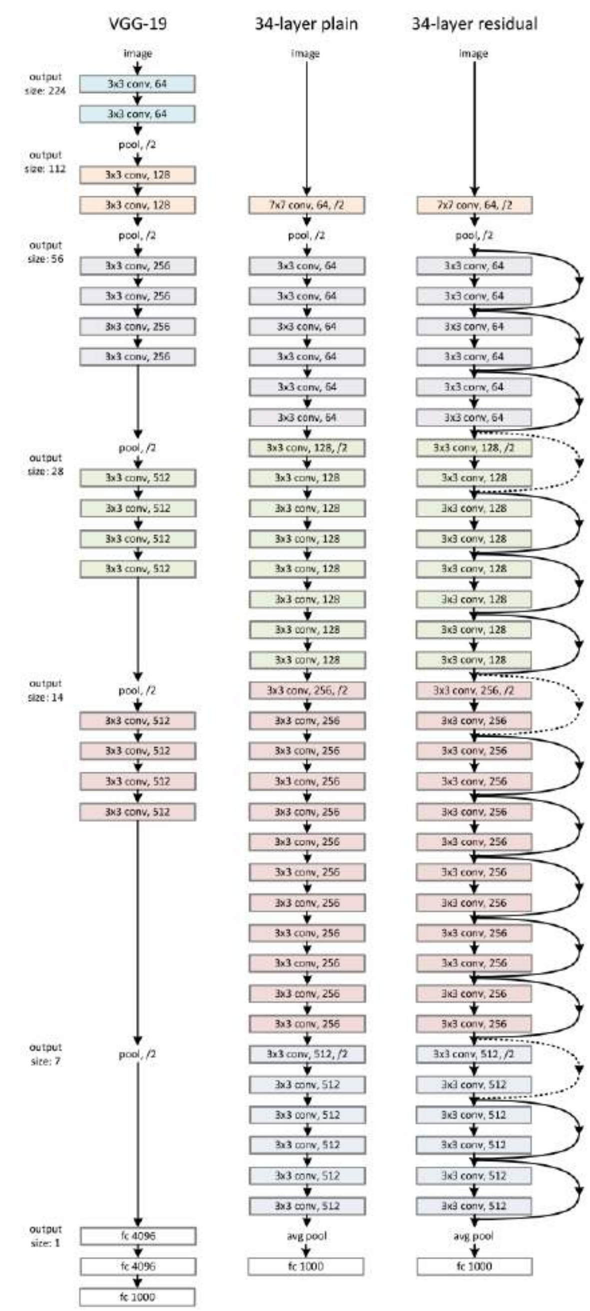
- Convolutional Neural Networks (CNN): AlexNet [2012], VGG [2014], ResNet [2016], …



AlexNet [packtpub.com]



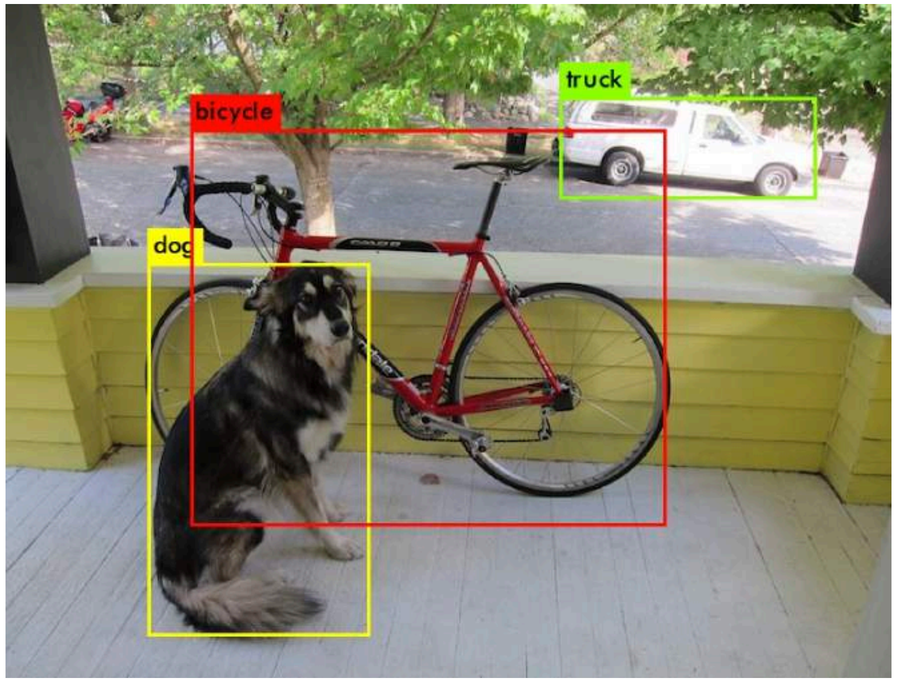
VGG16 [neurohive.io]



ResNet

Deep learning methods

- Networks specialized for the object detection problem: R-CNN, SPP-net, YOLO, …



YOLO output [pjreddie.com]

- Recent adaptations to embedded platforms: Tiny-YOLO, MobileNet, …
  - Smaller model size
  - Faster operation

Deep learning methods

- Advantages:
  - Very high performance
  - State of the art
- Disadvantages: Complex models
  - Need **data** to train
  - Need **time** to train
  - Very deep architectures may be **slow** to run on embedded systems

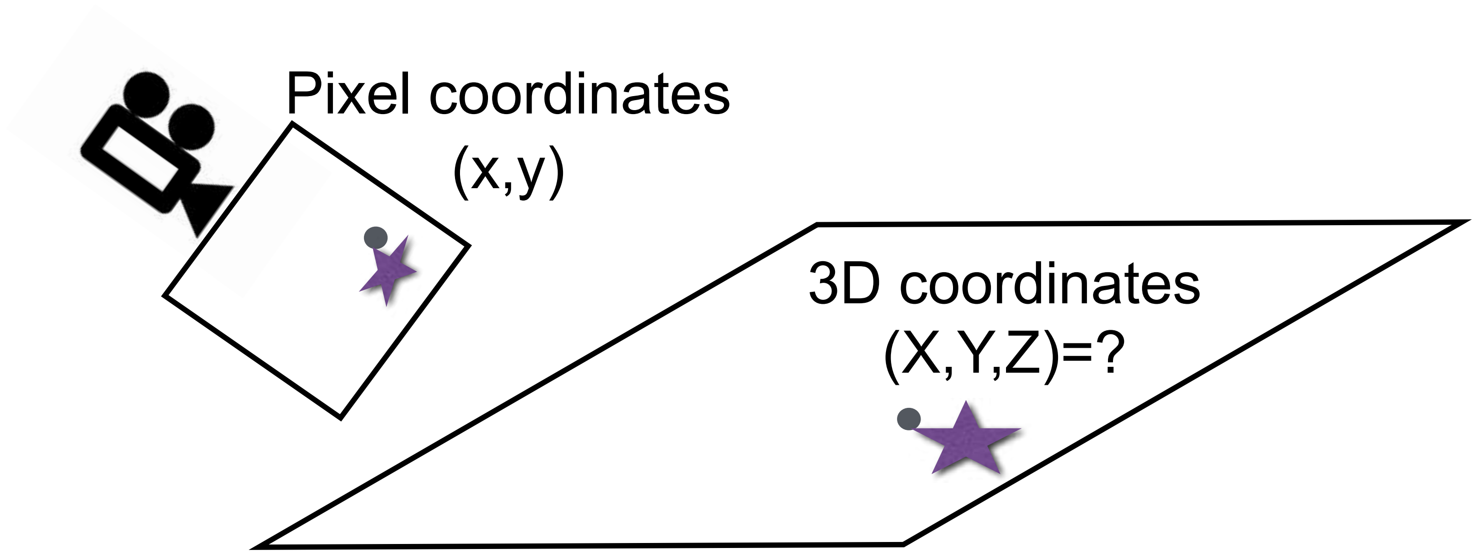
Solution:
- Take a pre-trained network
- Fine-tune final layers to adapt the network to your application

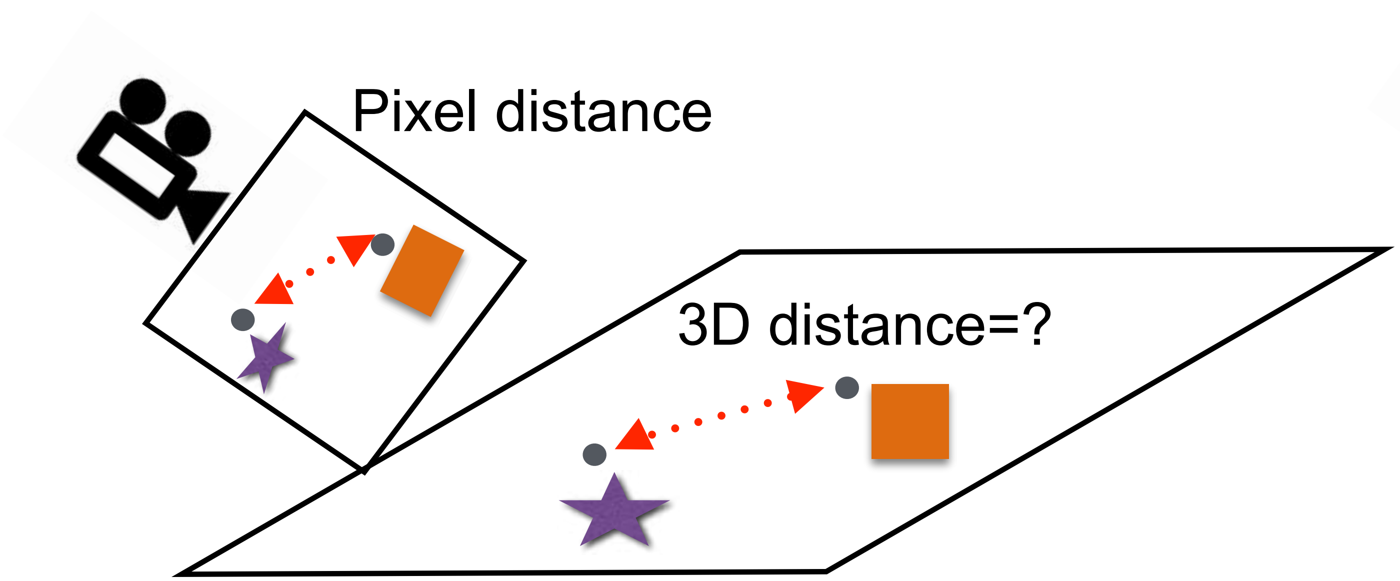
Outline

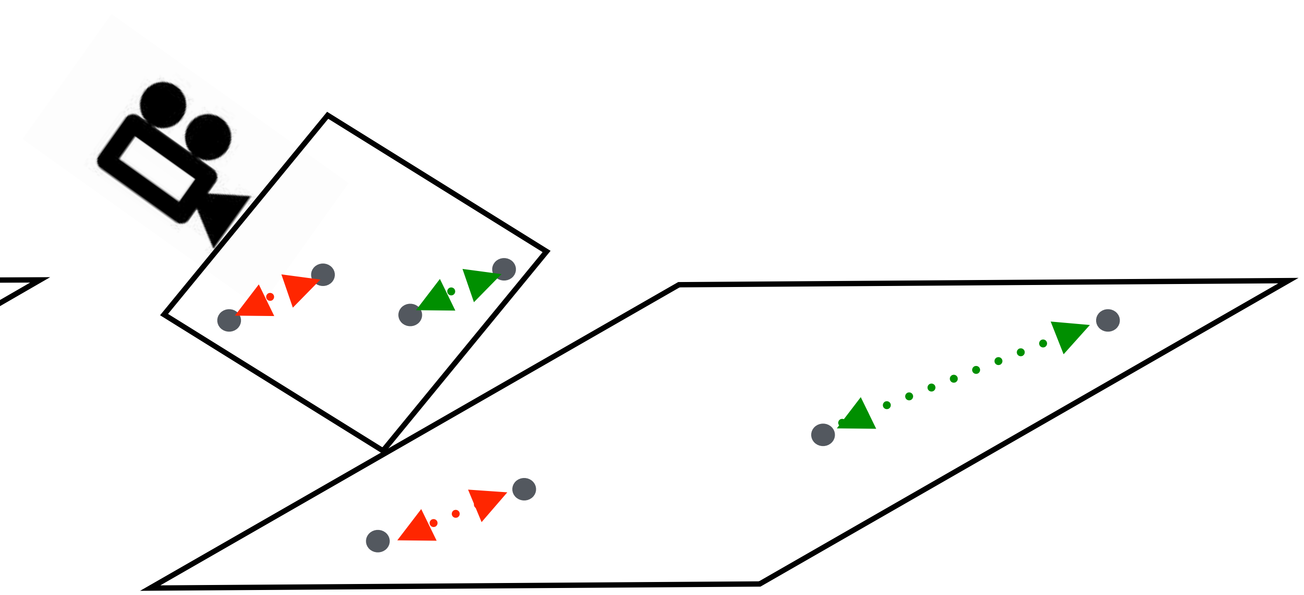
- Digital image representations
- Object and pattern detection
- 3D geometry and perspective correction

3D Geometry and Perspective Correction

- Problem: When capturing a scene, how to relate observed 2D pixel coordinates to actual 3D coordinates?

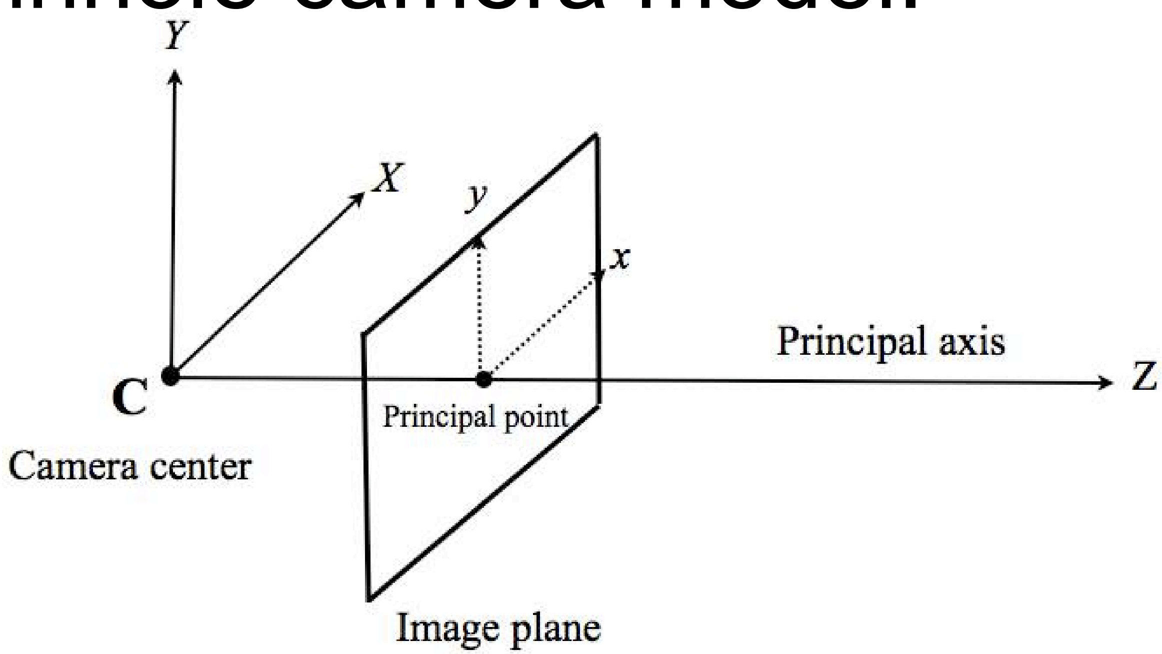






Relation Between 3D - 2D Coordinates

- Pinhole camera model:



- Derive the 2D coordinates of the image of a 3D point:

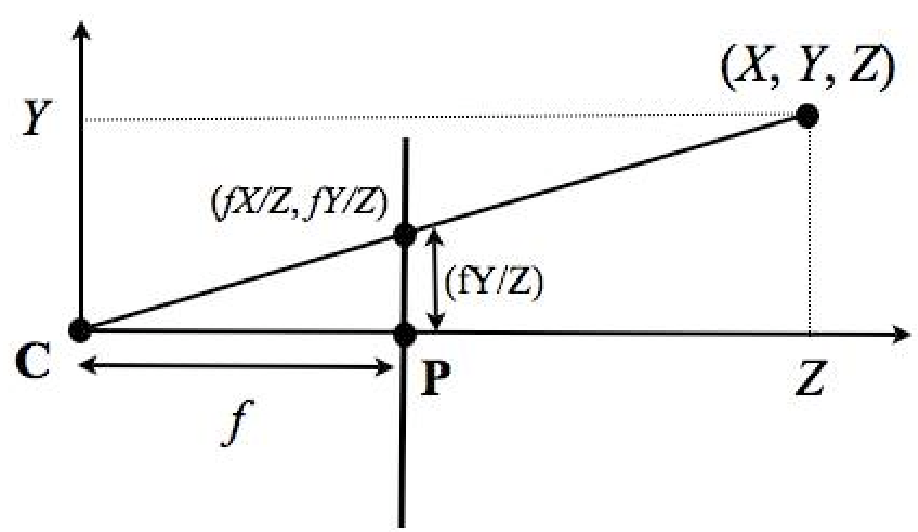


Image plane

The 3D point (X, Y, Z) is mapped to the 2D point $\left(f\frac{X}{Z}, f\frac{Y}{Z}\right)$

Pinhole Camera Model

3D point 2D projection Pixel coordinates

$$(X, Y, Z) \rightarrow \left(f\frac{X}{Z}, f\frac{Y}{Z} \right) \rightarrow \left(f\frac{X}{Z} + p_x,\ f\frac{Y}{Z} + p_y \right)$$

- Relation between 3D point and 2D point

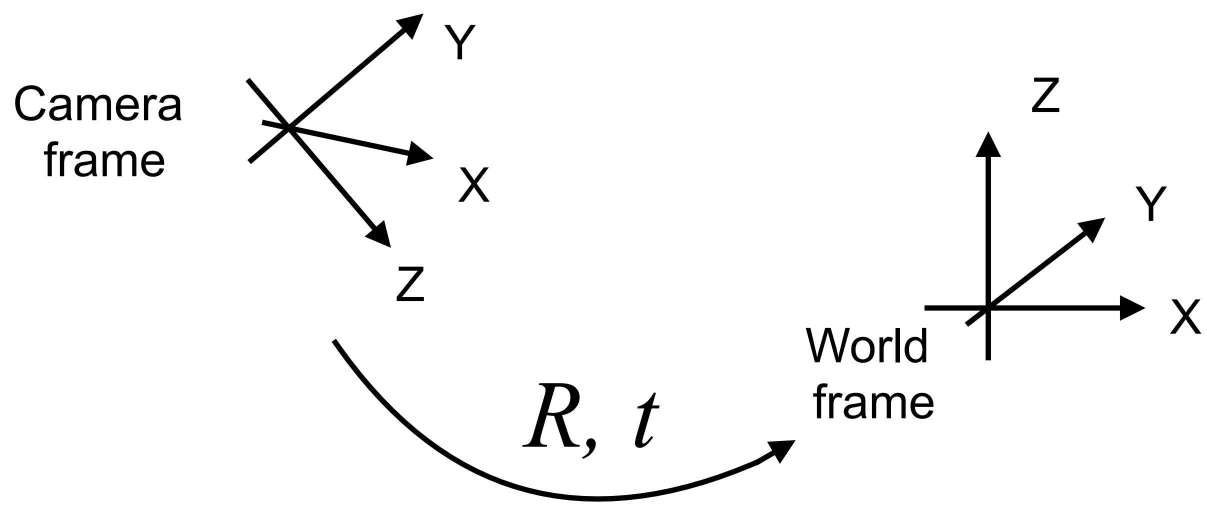
$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Homogeneous coordinates:

$$\underbrace{\begin{bmatrix} kx \\ ky \\ k \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{X}}$$

Pinhole Camera Model

- The camera frame is not necessarily aligned with the world frame



$$\mathbf{x} = [K|0]\mathbf{X}$$

Under rotation and translation

$$\mathbf{X} \rightarrow [R|t]\mathbf{X}$$

Pinhole camera model

$$\mathbf{x} = K[R|t]\mathbf{X}$$

$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} \qquad \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

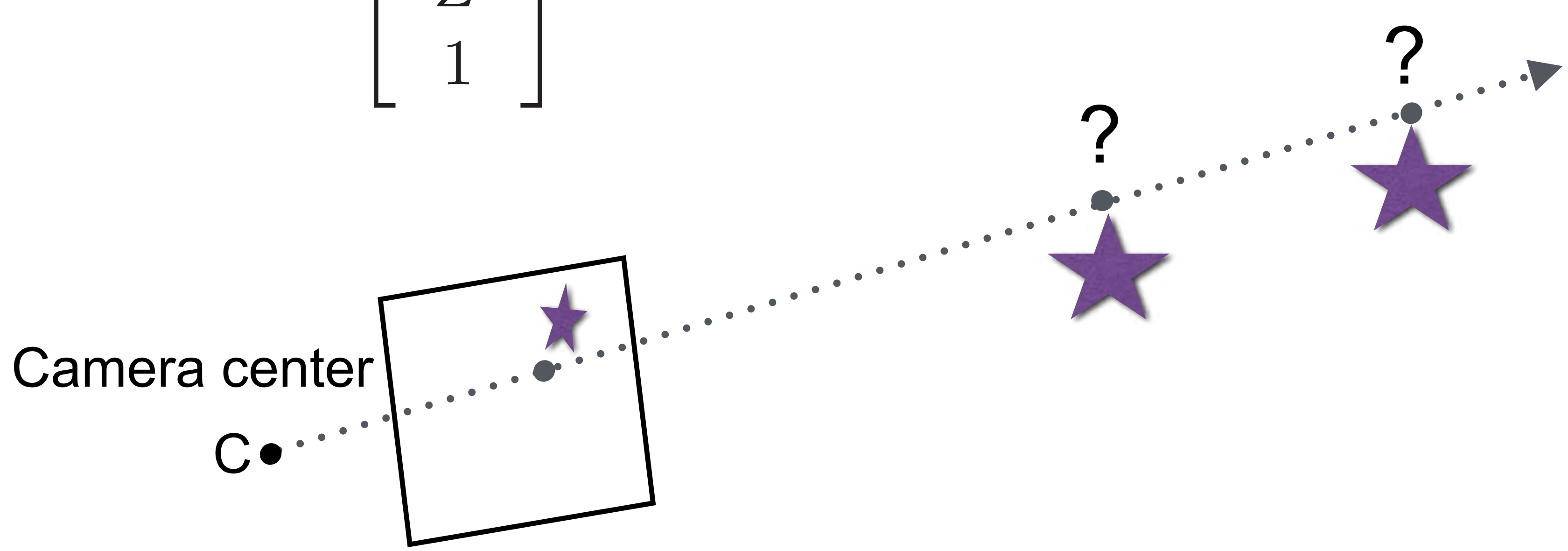
Is 3D Reconstruction Possible?

Pinhole camera model

$$\mathbf{x} = \underbrace{K[R|t]}_{\substack{\text{3x4 matrix} \\ \text{Not invertible!}}}\mathbf{X}$$

$$\mathbf{x} = \begin{bmatrix} kx \\ ky \\ k \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

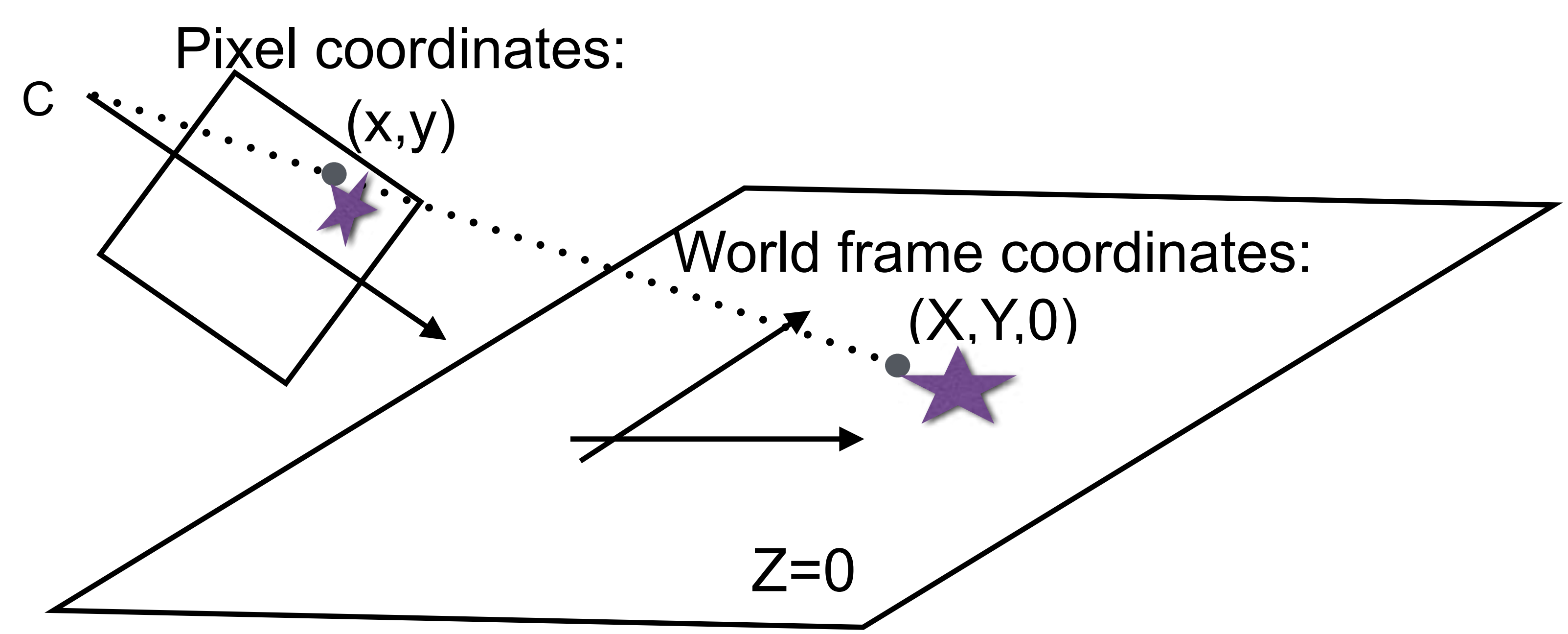
We cannot recover **X** from **x** in general.



- Good news: We can recover X when we know that it is on a planar surface!

Model Under Planar Scene

- Planar scene assumption: Let us take Z=0



$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \boxed{r_{13}} & t_x \\ r_{21} & r_{22} & \boxed{r_{23}} & t_y \\ r_{31} & r_{32} & \boxed{r_{33}} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ \boxed{0} \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Homographies

$$\mathbf{x} = \mathbf{H}\mathbf{X}$$

$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} [3 \times 3] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

- This relation is called a homography.
- H is a 3x3 invertible matrix.
- The 3D point X can be recovered from its pixel coordinates!

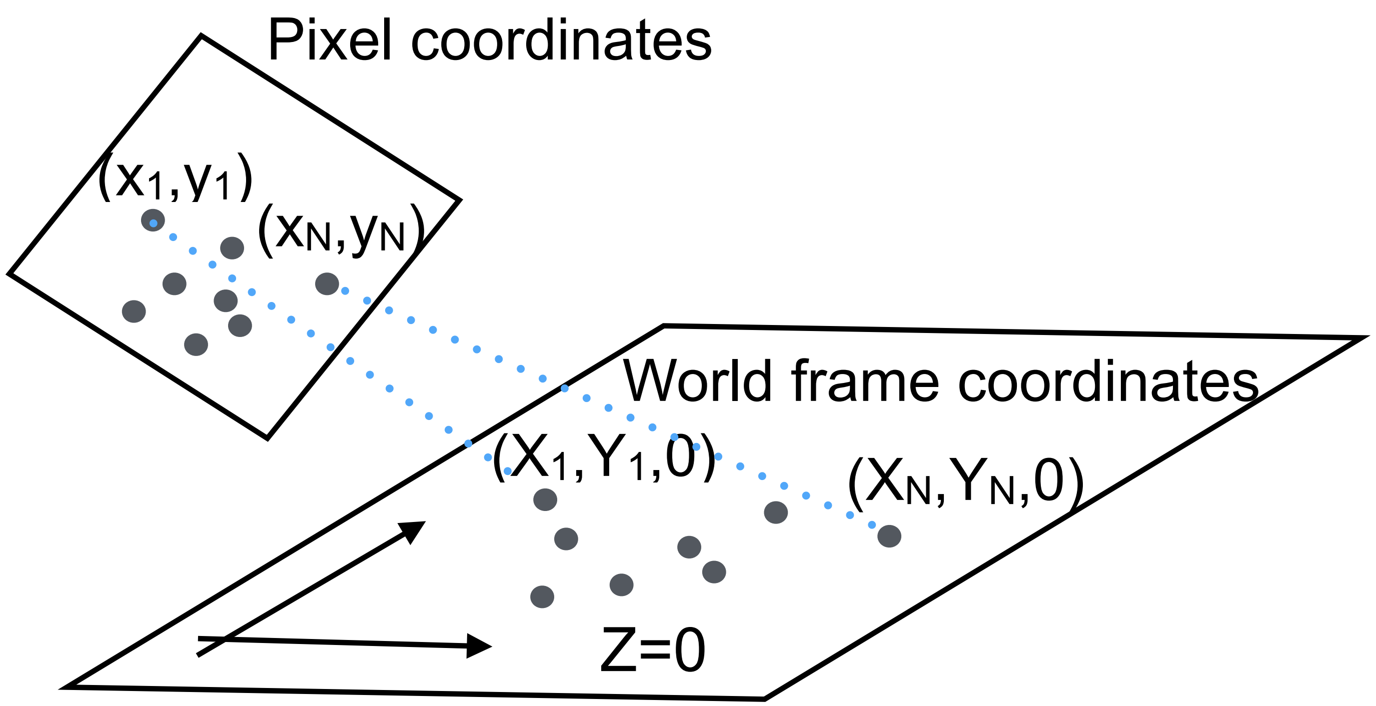
How to correct the perspective distortion for planar points:

1. Compute the homography matrix from a set of known 3D points on a plane and their pixel coordinates

2. Find the matrix H^{-1}

3. Given **x** in pixel coordinates, find the 3D point as $\mathbf{X} = \mathbf{H}^{-1}\,\mathbf{x}$

Computing the Homography



$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & \underbrace{\boxed{h_{33}}}_{=1} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

- Taking $h_{33}=1$ for normalization, the relation $\mathbf{x_i}= \mathbf{H}\ \mathbf{X_i}$ gives

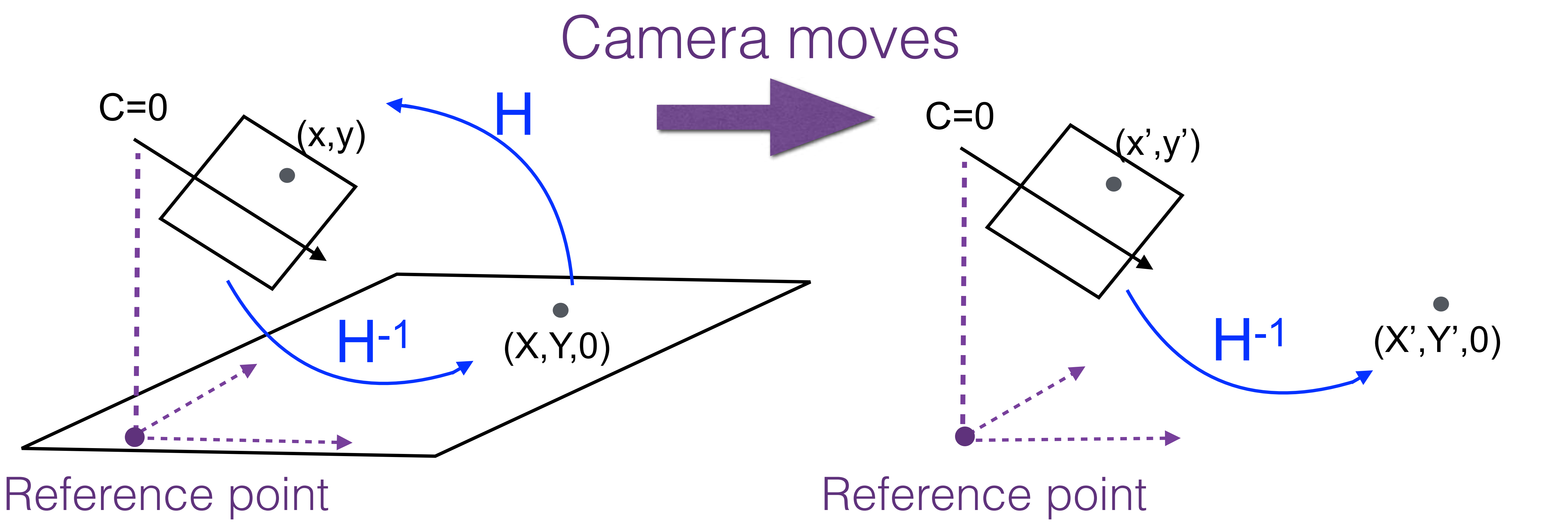
$$x_i = \frac{h_{11}X_i + h_{12}Y_i + h_{13}}{h_{31}X_i + h_{32}Y_i + 1} \quad , \quad y_i = \frac{h_{21}X_i + h_{22}Y_i + h_{23}}{h_{31}X_i + h_{32}Y_i + 1}$$

- N such 2D-3D point matches gives 2N equations in unknowns $\{h_{11}, h_{12}, h_{13}, \ldots, h_{32}\}$

Computing the Homography

- Form a linear equation system and solve for the unknown homography parameters $\{h_{11}, h_{12}, h_{13}, \ldots, h_{32}\}$
- Warning: Too large pixel coordinates may cause numerical instability!
  - Normalize the coordinates to 0-mean and an average norm of sqrt(2)
  - Compute the homography parameters
  - Undo the normalization

Perspective Correction



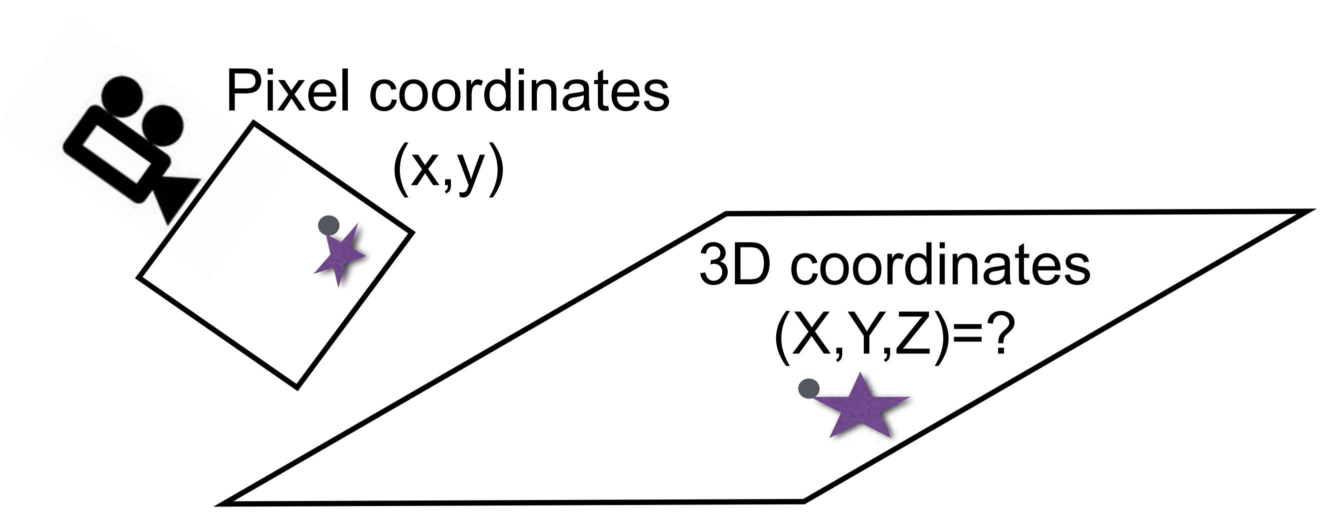
- Let the angle between the **camera frame and the plane** be fixed:
- Then even if the camera moves, through H^{-1} we can get the relative coordinates (X', Y', 0) with respect to the camera.

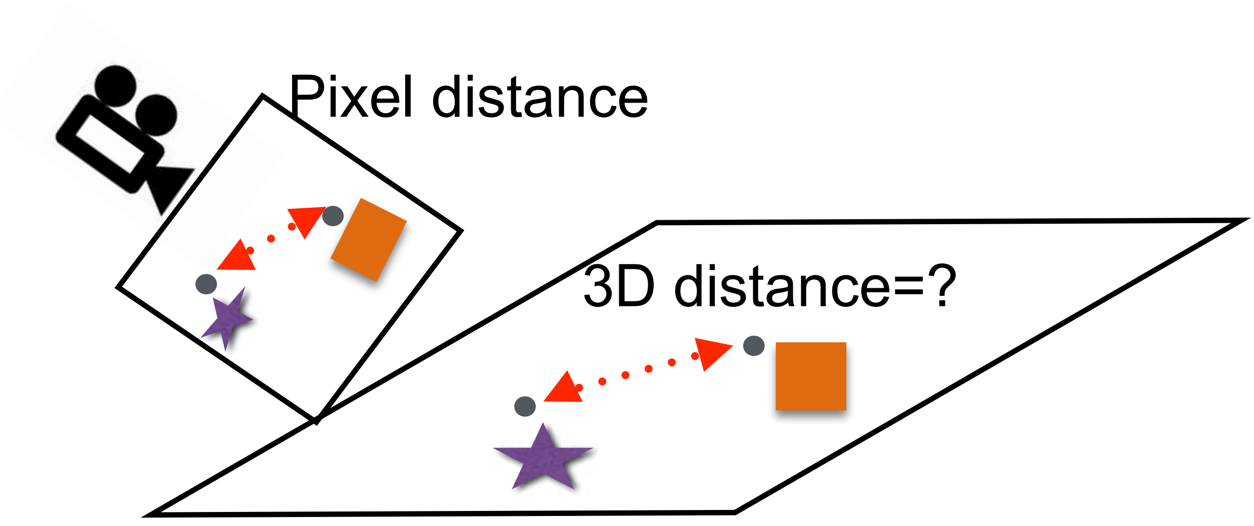
Conclusions

- Object detection
  - Shape priors
  - Template matching, feature detection
  - Deep learning: Needs data and time

Conclusions

- Perspective correction problem:





- Easy to do if the scene is planar and camera looks at the scene from a constant angle
- Learn a homography model!