Principles of Robot Autonomy I

Information extraction

Stanford University

Autonomous Systems Lab

Agenda

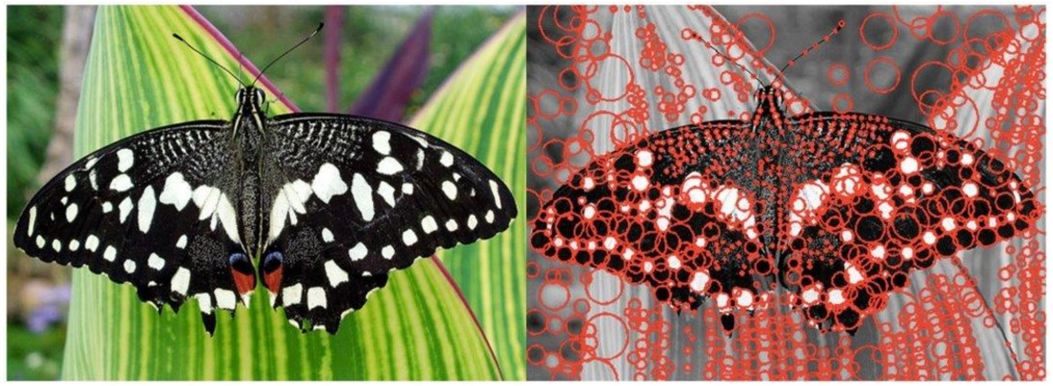
- Agenda
  - Introducing SiFT
  - Extracting information from sensor measurements

- Readings:
  - Chapters 11 in PoRA lecture notes

Last lecture: Recap

- Image processing, feature detection and description, such as:
  - Correlation / convolution filtering operations (left figure)
  - Feature descriptors for detecting salient keypoints (right figure)

Canny edge detector
(filter + convolution)

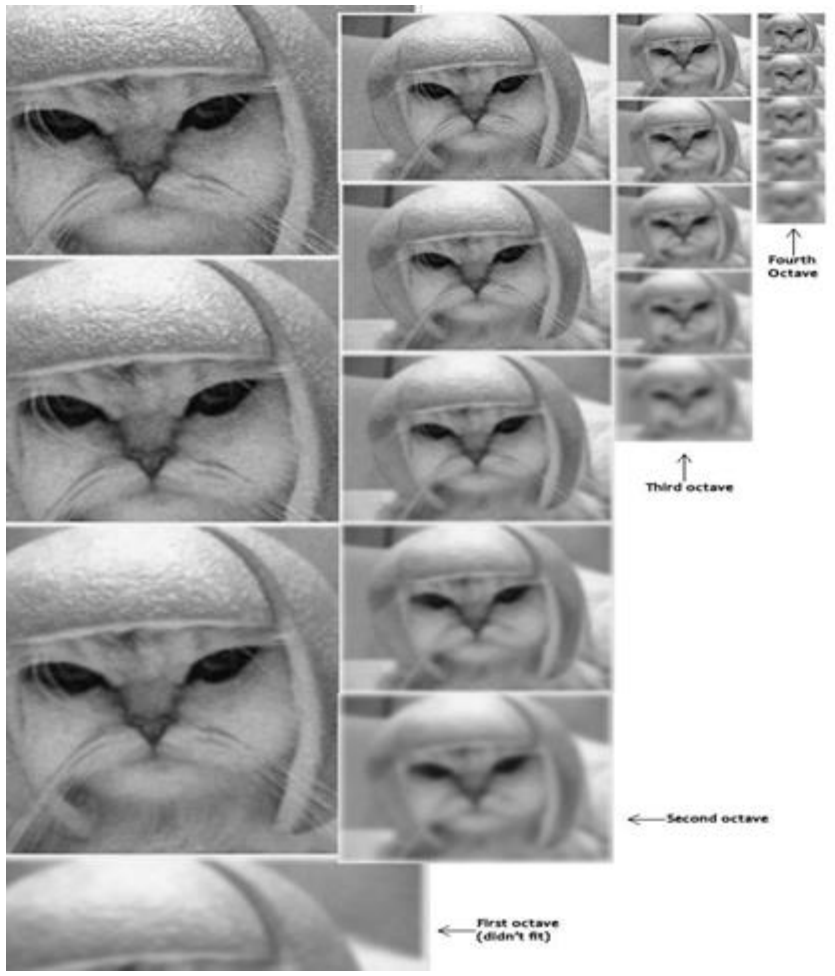


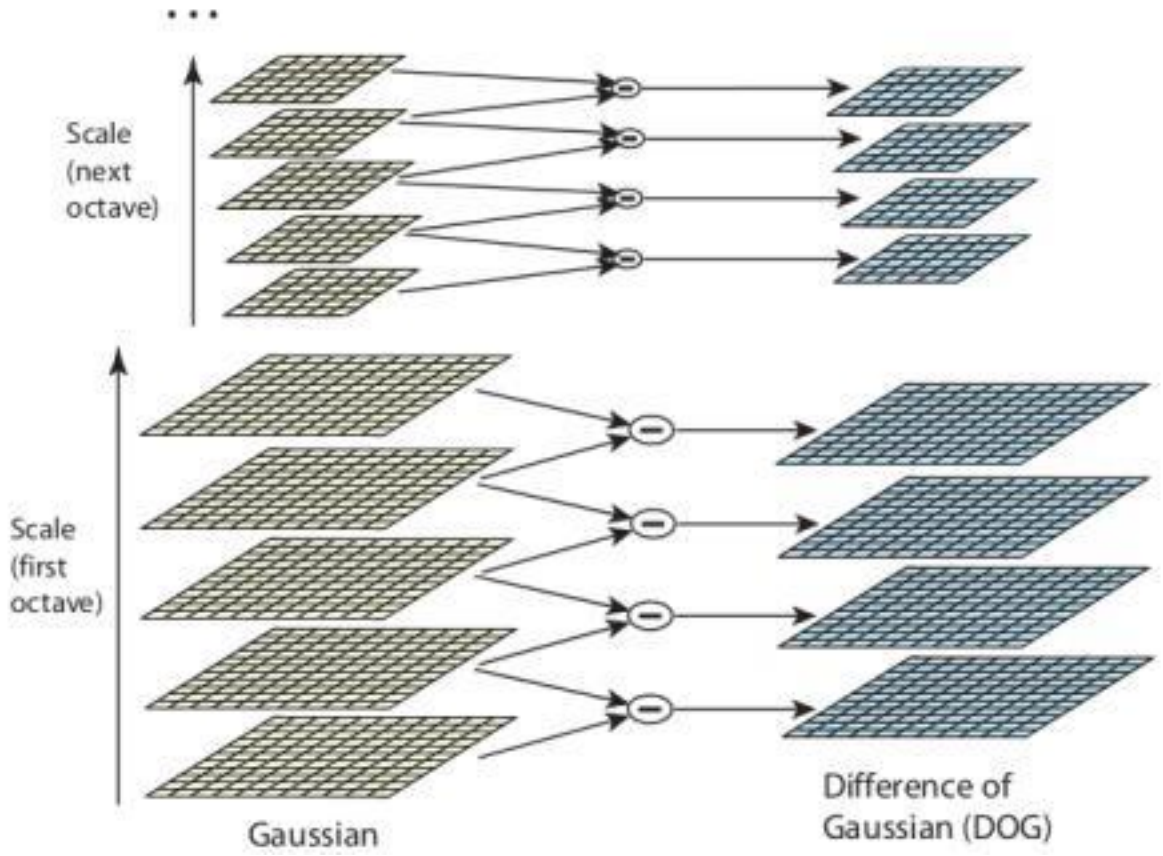
Keypoints from e.g., SIFT

Recap Feature Detection

1. Repeatability: same feature can be found in multiple images despite geometric and photometric transformations
2. Distinctiveness: information carried by the patch surrounding the feature should be as distinctive as possible

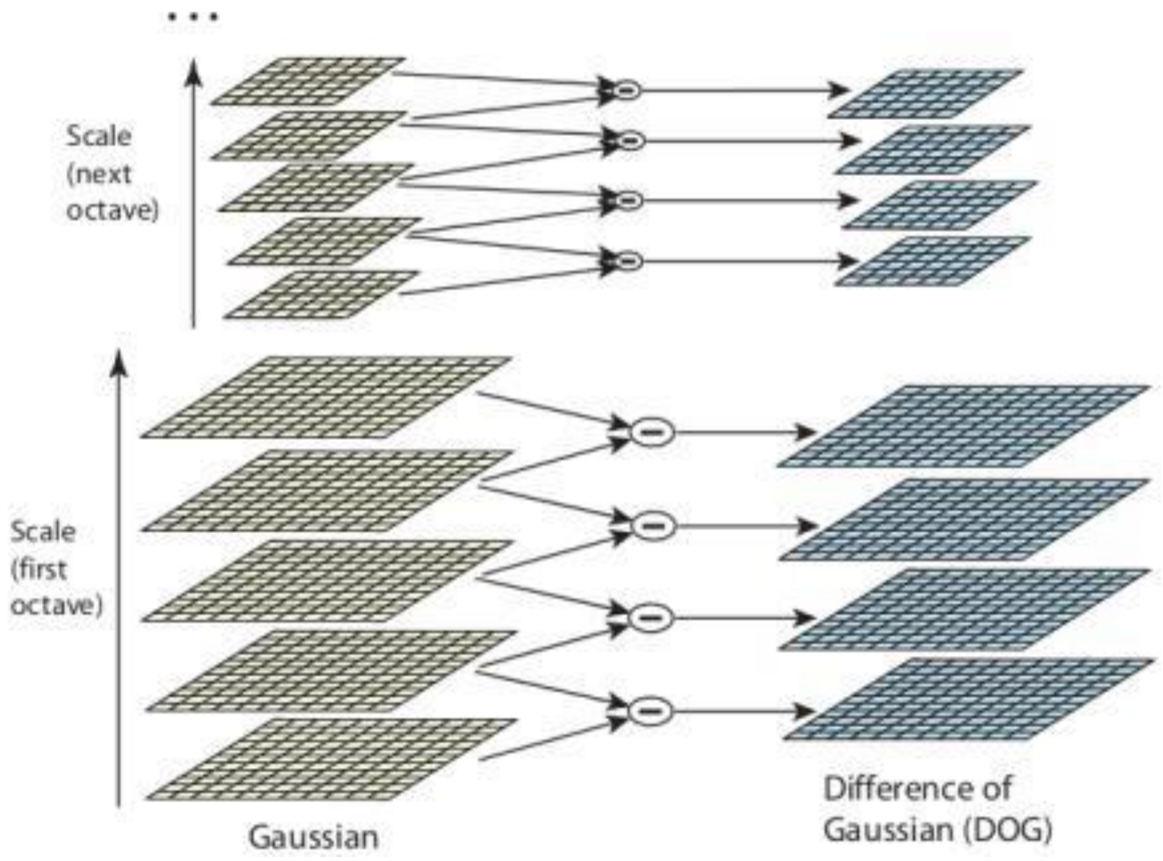
Introducing SiFT Detector

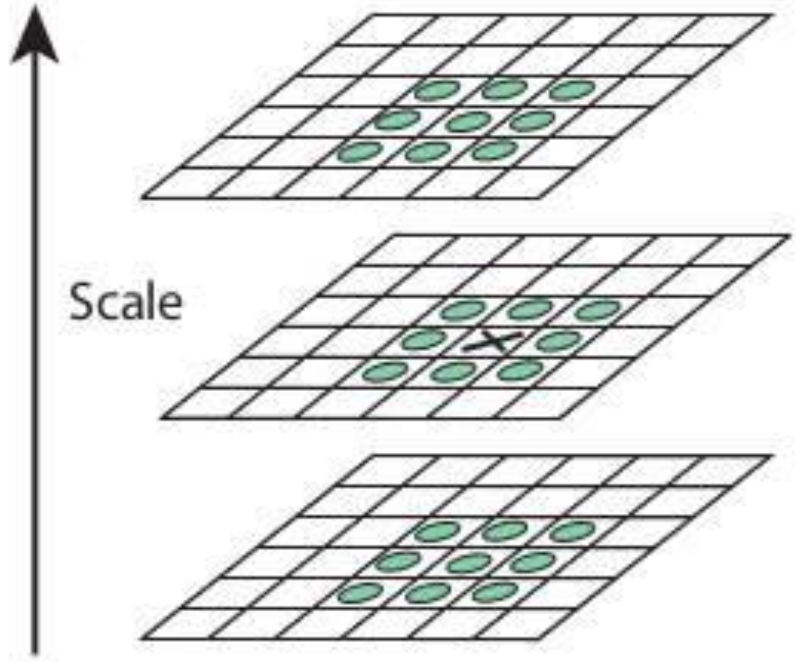




https://www.youtube.com/watch?v=4AvTMVD9ig0
https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

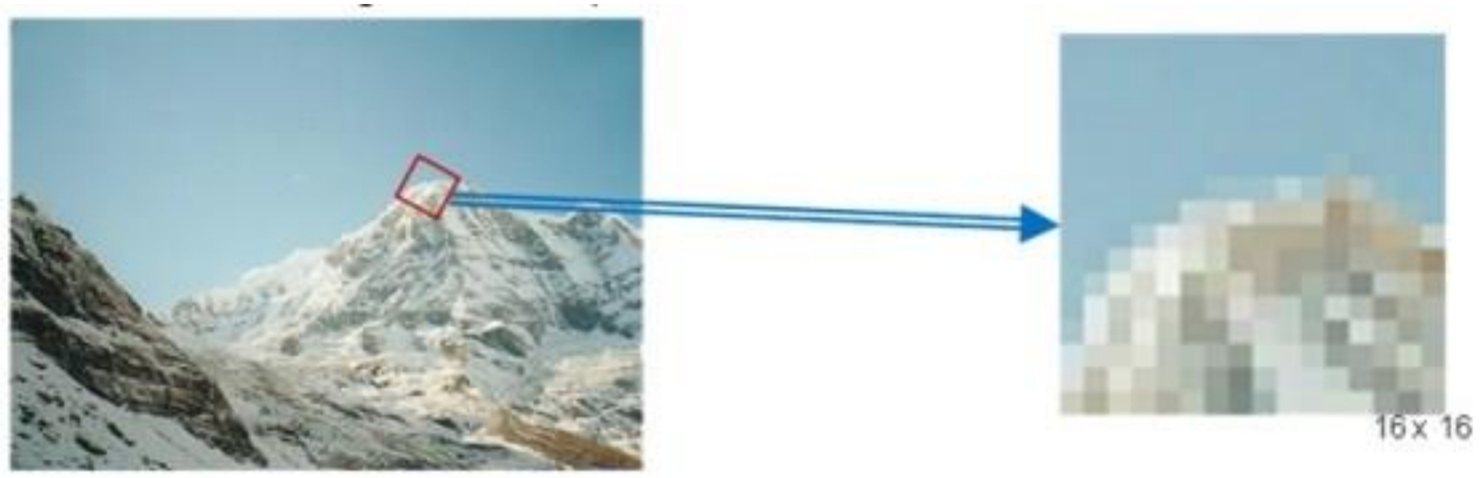
Introducing SiFT Detector

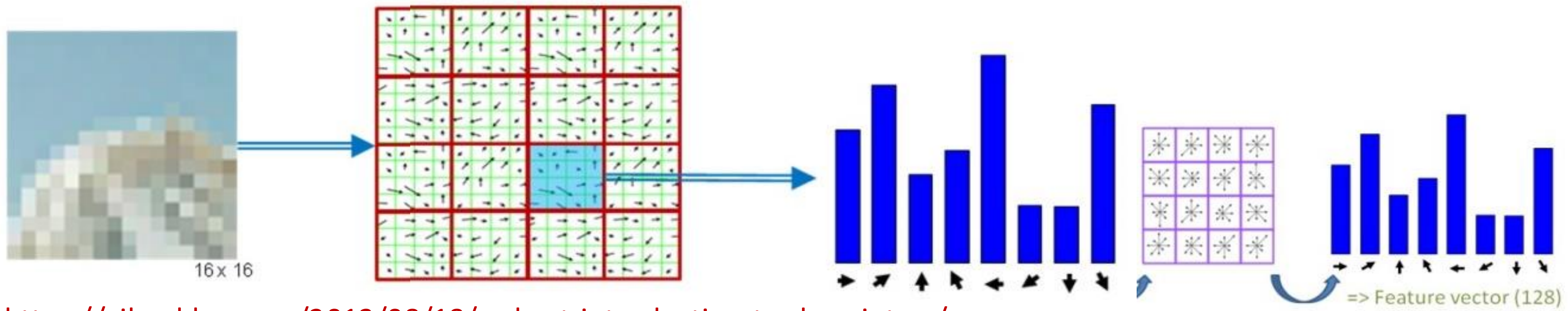




https://www.youtube.com/watch?v=4AvTMVD9ig0
https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

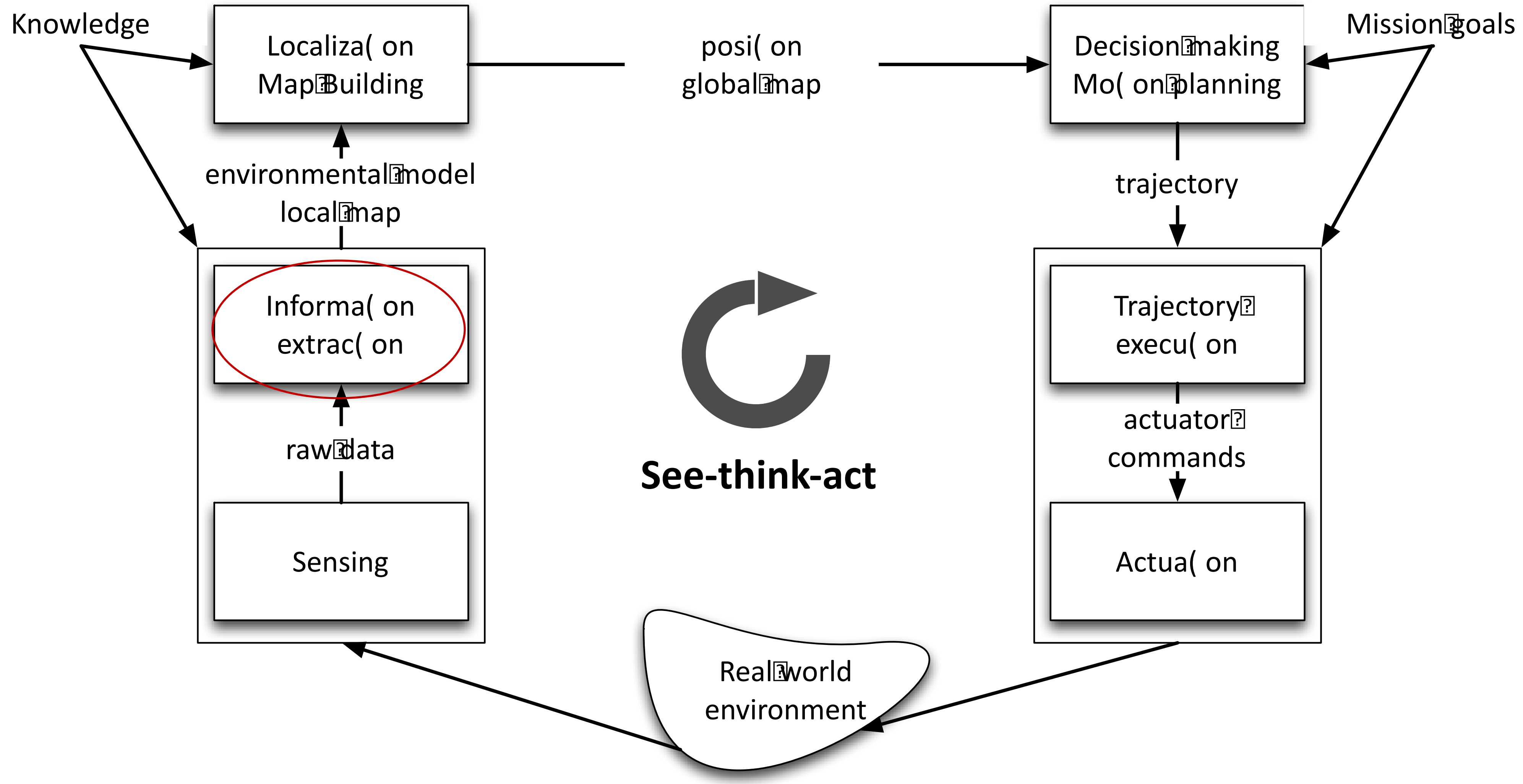
Introducing SiFT Descriptor





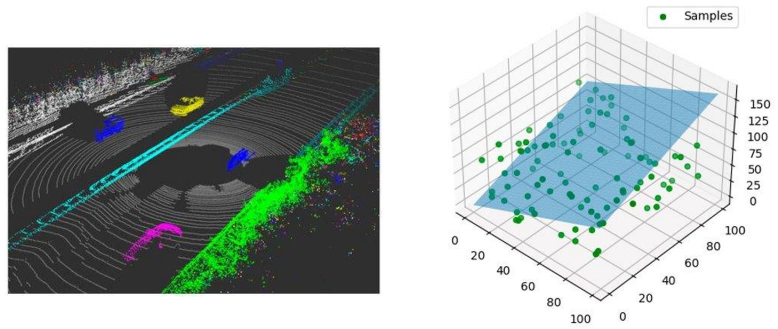
https://gilscvblog.com/2013/08/18/a-short-introduction-to-descriptors/

The see-think-act cycle

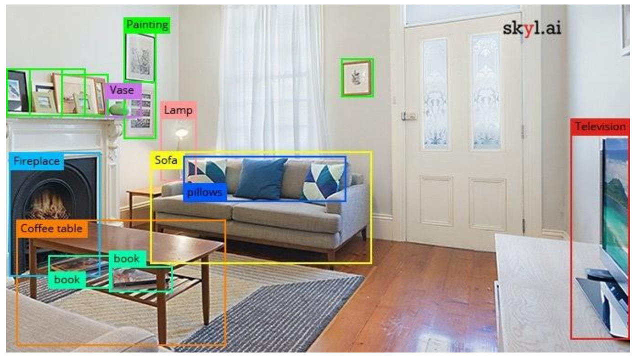


Information extraction

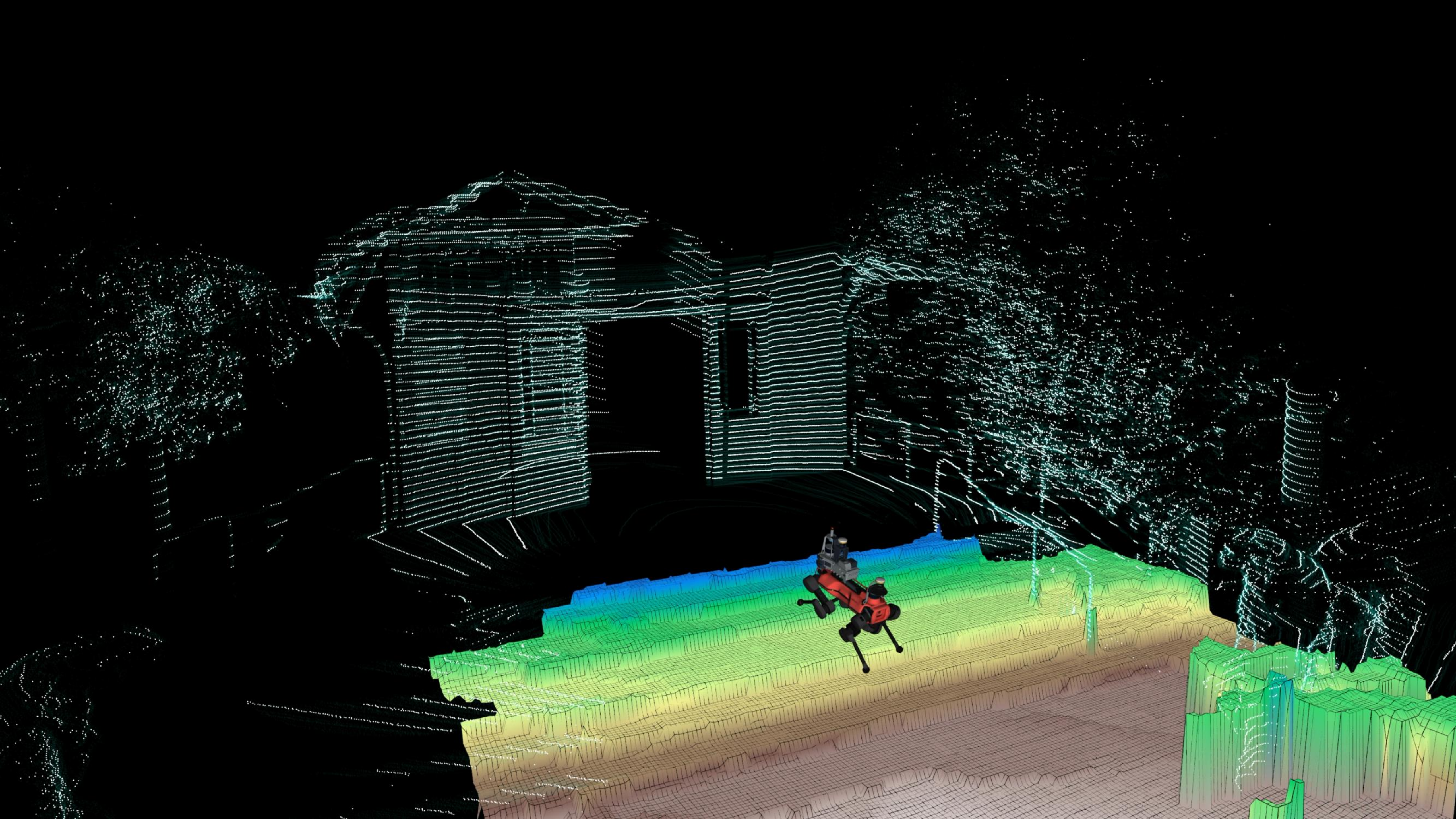
- Today's focus: extracting *actionable information* from images
    1. Geometric primitives (e.g., lines and circles): useful, for example, for robot localization and mapping
    2. Scene understanding and object recognition: useful, for example , for localization within a topological map and for high-level reasoning



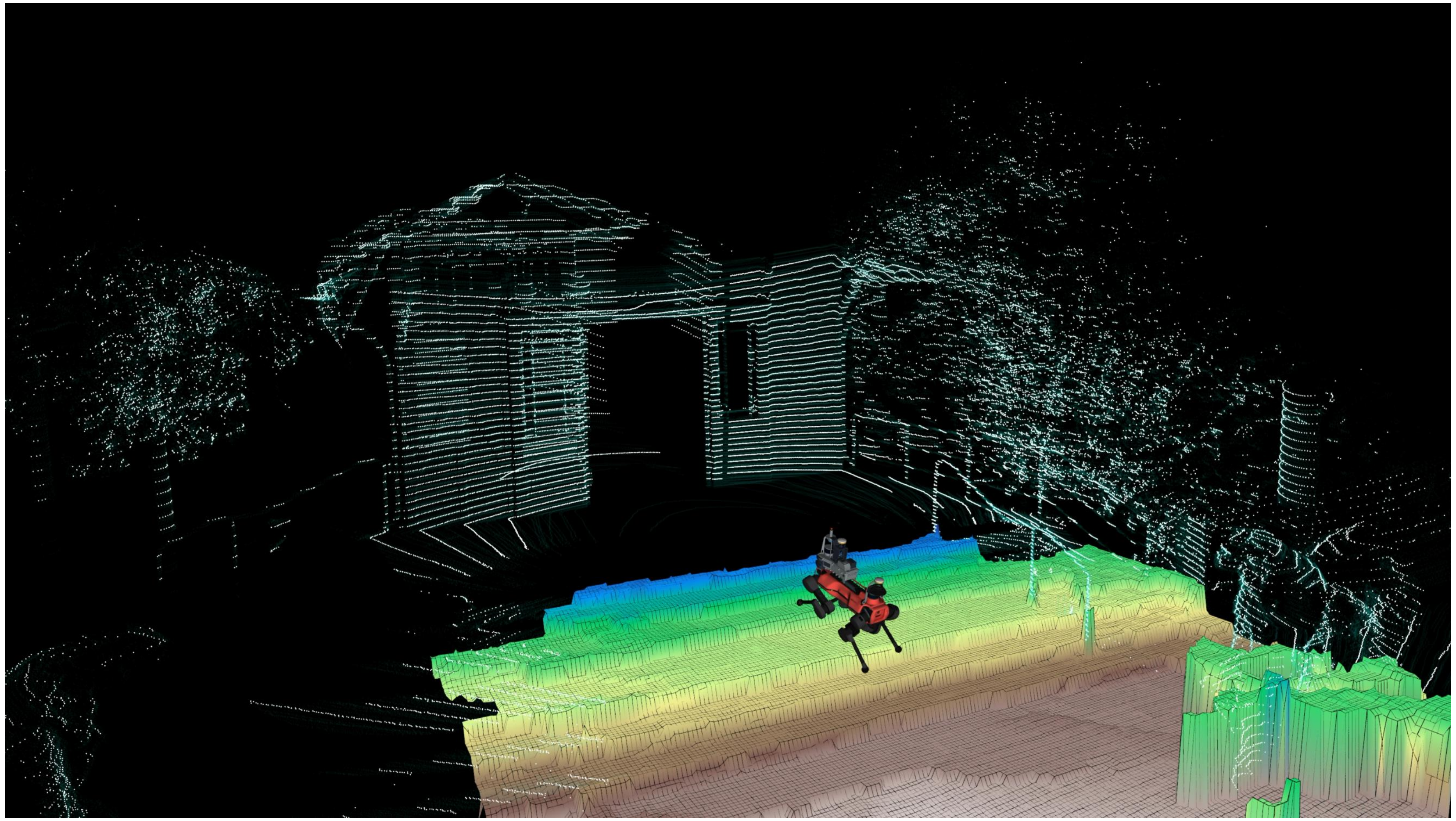
Example (Geometric primitive):
Plane Fitting



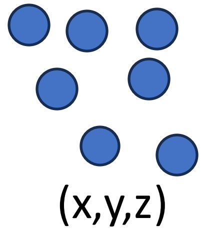
Example (Scene understanding):
Object detection



See – Think – Act



1. Our robot sees points



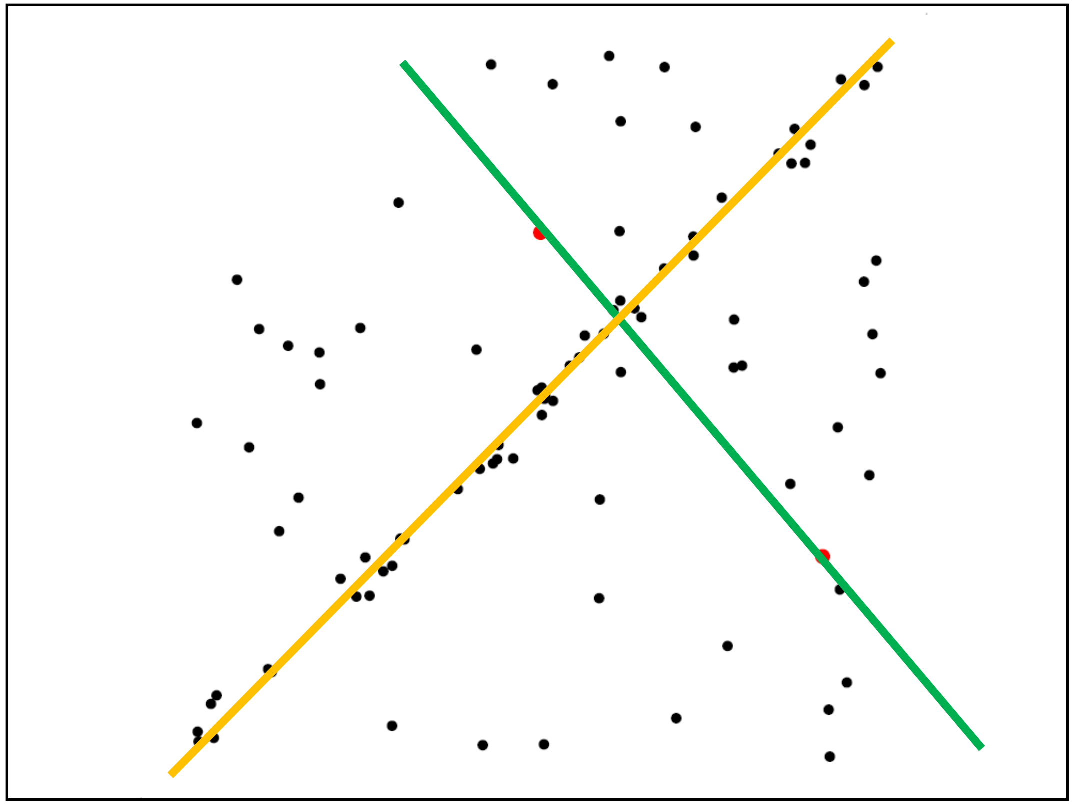
2. We fit an elevation map to the data

3. We act based on the map representation

Geometric information extraction

- Geometric feature extraction: extract geometric primitives from sensor data (e.g., range data)
- Examples: line, circles, corners, planes, etc.
- We focus on *line extraction* from range data (a quite common task); other geometric feature extraction tasks are conceptually analogous
- The two main problems of line extraction from range data
  1. Which points belong to which line? → *segmentation*
  2. Given an association of points to a line, how to estimate line parameters? → *fitting*

Intuition Line Fitting



Step #2: line fitting

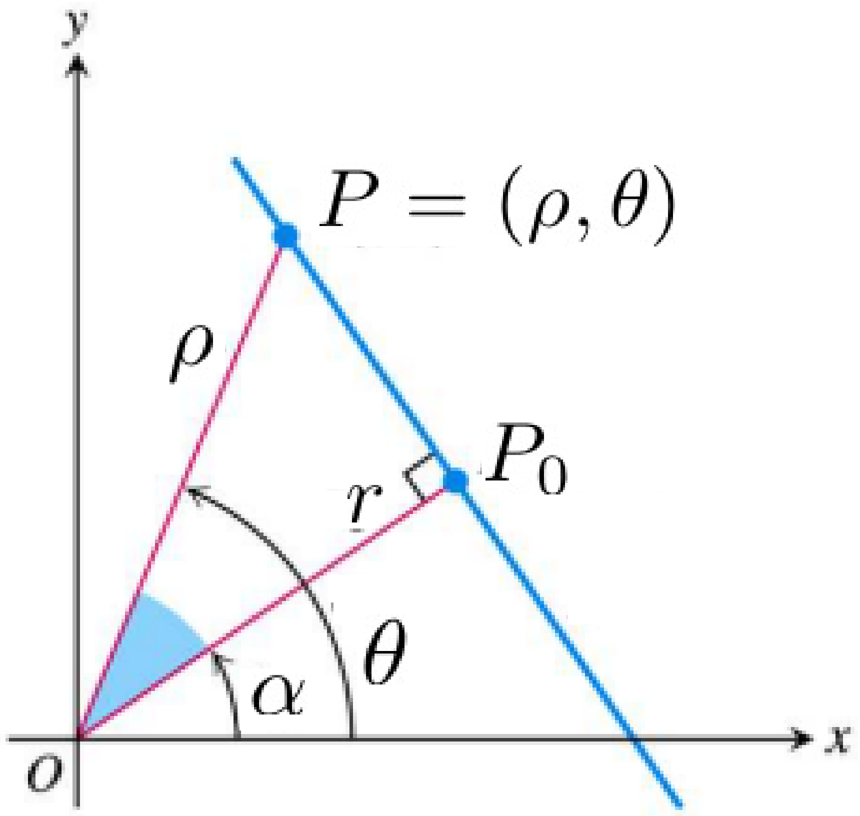
- Goal: fit a line to a set of sensor measurements
- It is useful to work in polar coordinates:

$$x = \rho\cos\theta, \qquad y = \rho\sin\theta$$

- Equation of a line in polar coordinates
  - Let $P = (\rho, \theta)$ be an arbitrary point on the line
  - Since P, P_0, O determine a right triangle

$$\boxed{\rho\cos(\theta - \alpha) = r}$$

- (r, α) are the parameters of the line



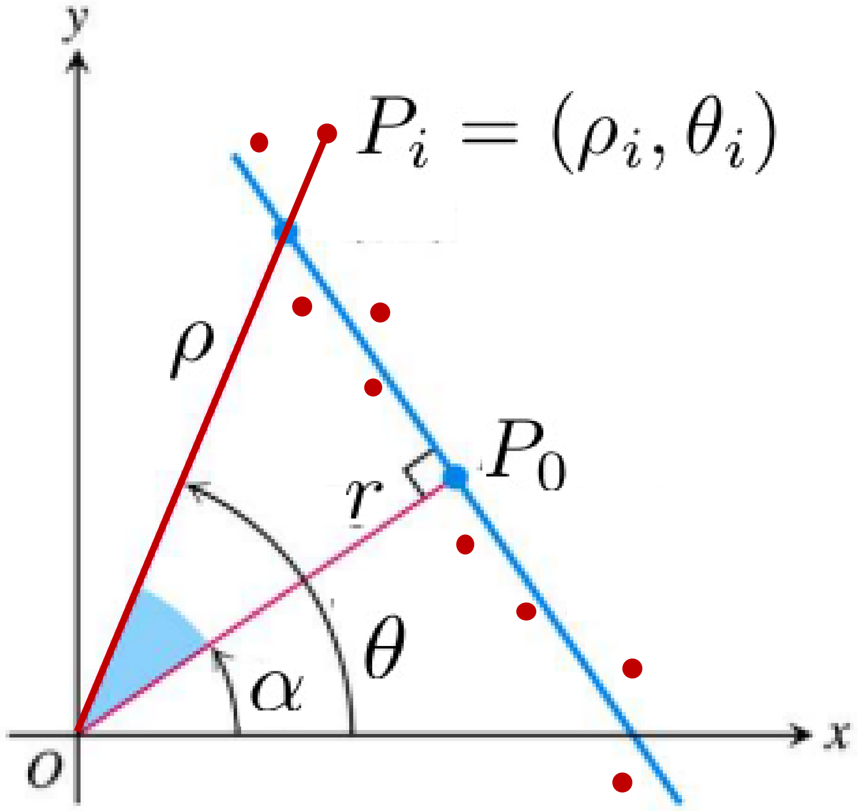
Step #2: line fitting

- Since there is measurement error, the equation of the line is only *approximately* satisfied

$$\rho_i \cos(\theta_i - \alpha) = r + d_i$$

Error

- Assume *n* ranging measurement points represented in polar coordinates as (ρ_i, θ_i)
- We want to find a line that best "fits" all the measurement points



Step #2: line fitting

- Consider, first, that all measurements are equally uncertain
- Find line parameters (r, α) that minimize squared error

$$S(r,\alpha) := \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Unweighted least squares

Step #2: line fitting

- Consider, now, the case where each measurement has its own, unique uncertainty
- For example, assume that the variance for each range measurement ρ_i is σ_i
- Associate with each measurement a weight, e.g., $w_i = 1/\sigma_i^2$
- Then, one minimizes

$$S(r, \alpha) := \sum_{i=1}^{n} w_i \, d_i^2 = \sum_{i=1}^{n} w_i \left(\rho_i \cos(\theta_i - \alpha) - r \right)^2$$

- Weighted least squares

Step #2: line fitting solution

- Assume that the *n* ranging measurements are independent
- Solution:

$$\alpha = \frac{1}{2}\mathrm{atan2}\left(\frac{\sum_i w_i\rho_i^2 \sin 2\theta_i - \frac{2}{\sum_i w_i}\sum_i\sum_j w_i w_j \rho_i \rho_j \cos\theta_i \sin\theta_j}{\sum_i w_i\rho_i^2 \cos 2\theta_i - \frac{1}{\sum_i w_i}\sum_i\sum_j w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)}\right) + \frac{\pi}{2}$$

$$r = \frac{\sum_i w_i \rho_i \cos(\theta_i - \alpha)}{\sum_i w_i}$$

Step #1: line segmentation

- Several algorithms are available
    1. Split-and-merge
    2. RANSAC
    3. Hough-Transform

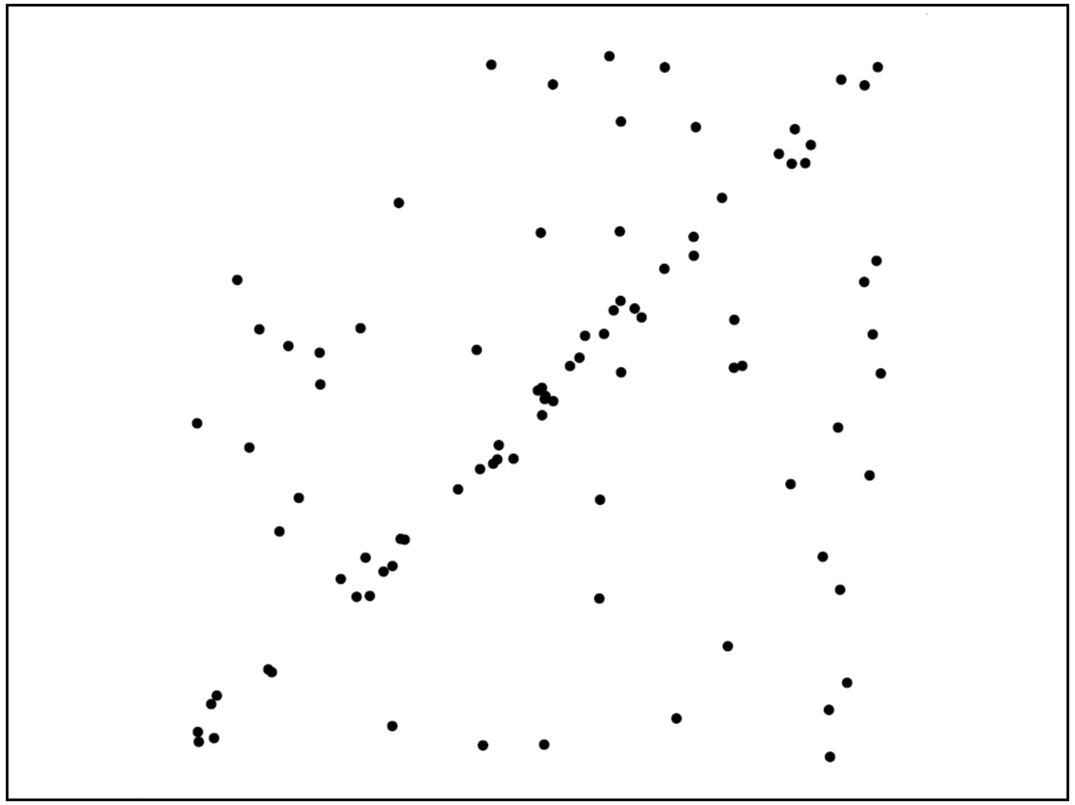
- We will focus on RANSAC

RANSAC

- RANSAC: **Ran**dom **Sa**mple **C**onsensus
- General method to estimate parameters of a model from a set of observed data in the presence of outliers, where outliers should have no influence on the estimates of the values
- Typical applications in robotics: line extraction from 2D range data, plane extraction from 3D point clouds, feature matching for structure from motion, etc.
- RANSAC is *iterative* and *non-deterministic*: the <u>probability</u> of finding a set free of outliers increases as more iterations are used

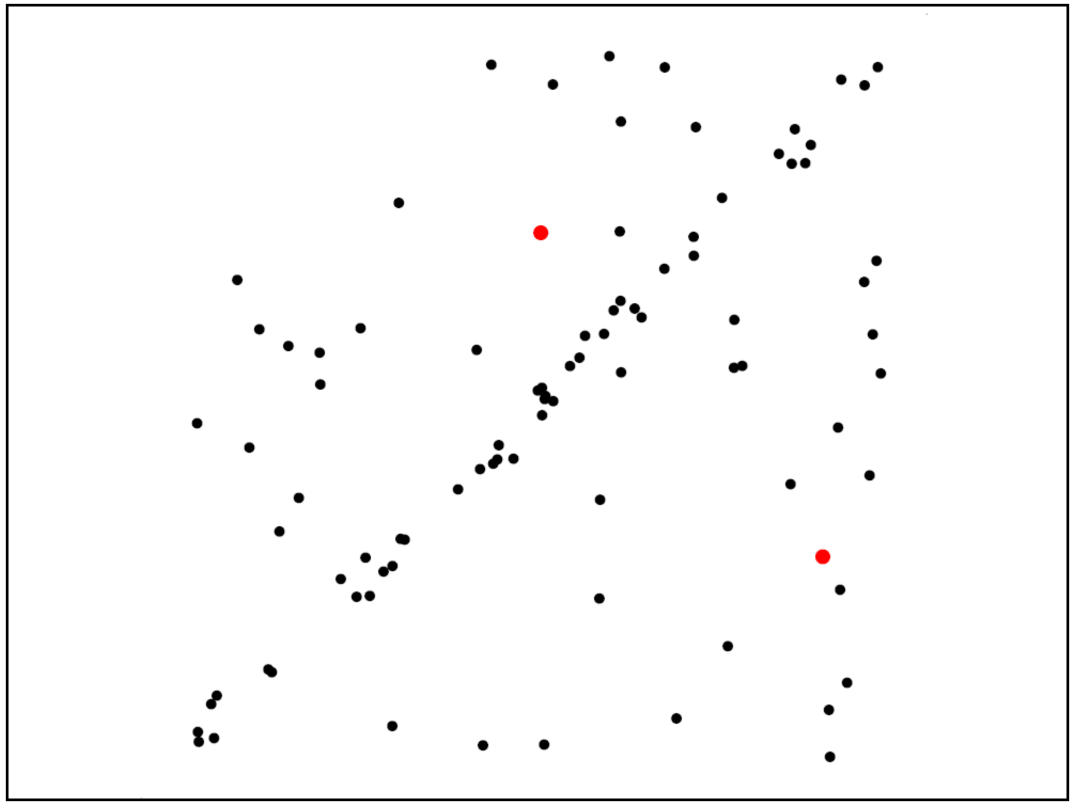
RANSAC

```
**Data:** Set $S$ consisting of all $N$ points
**Result:** Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



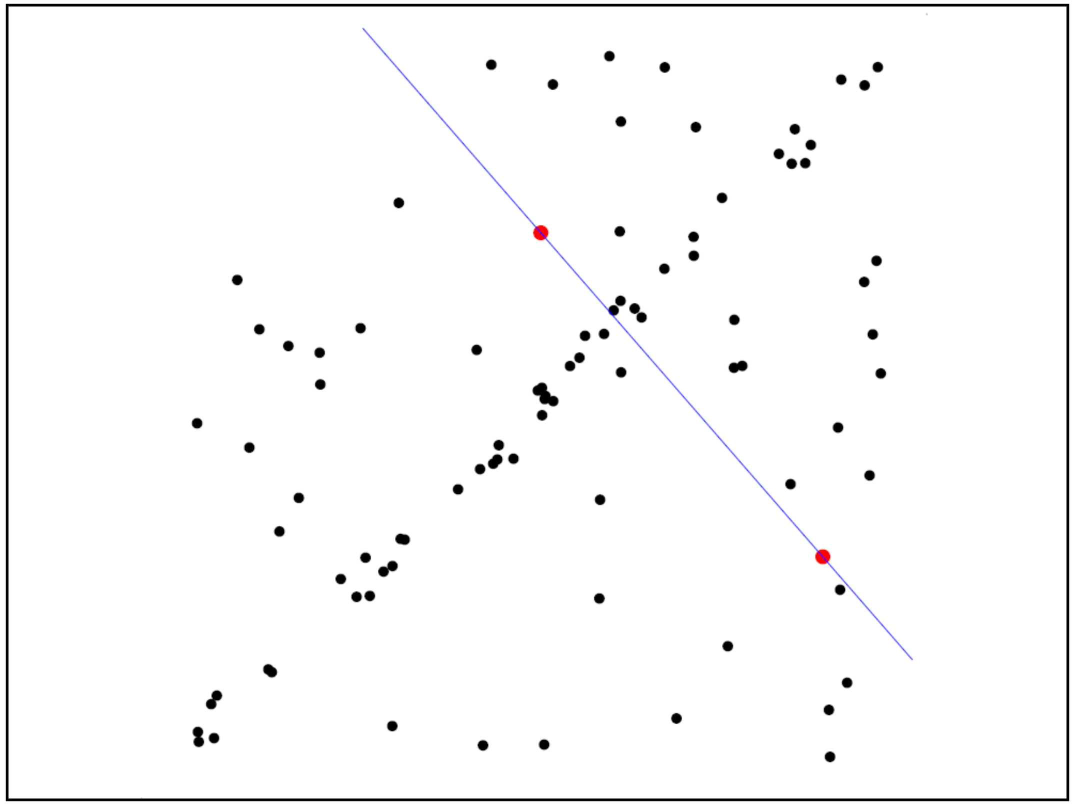
RANSAC

```
**Data:** Set $S$ consisting of all $N$ points
**Result:** Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



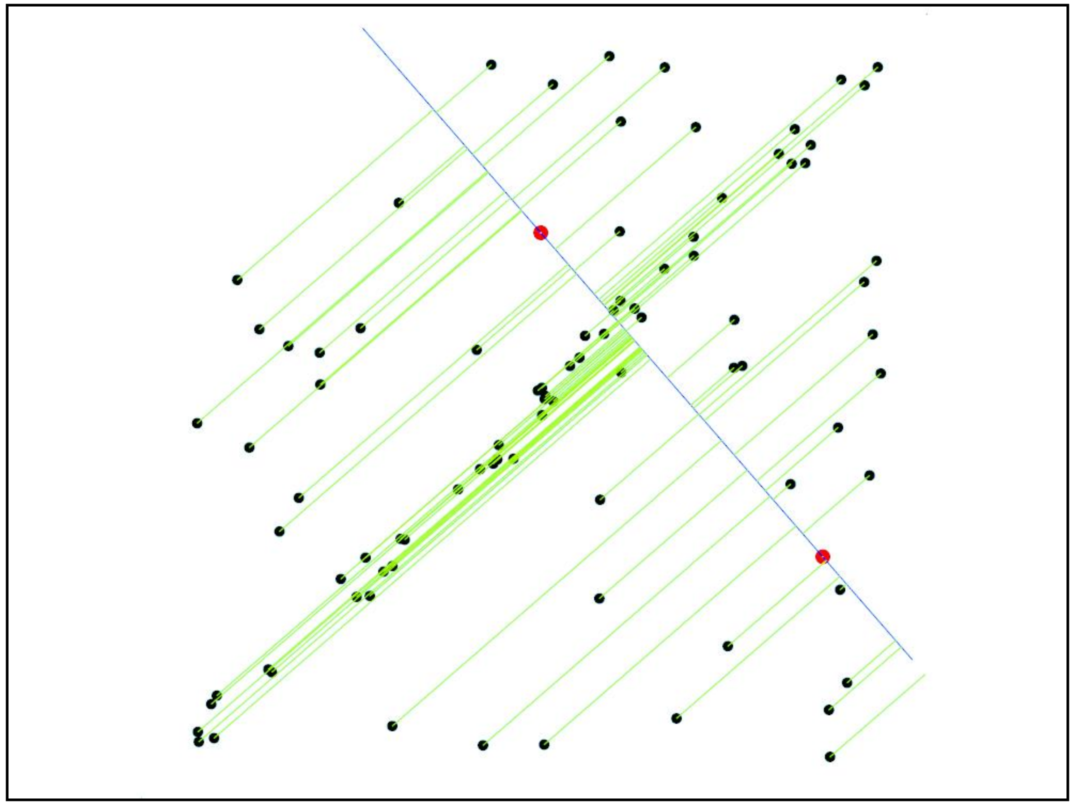
RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



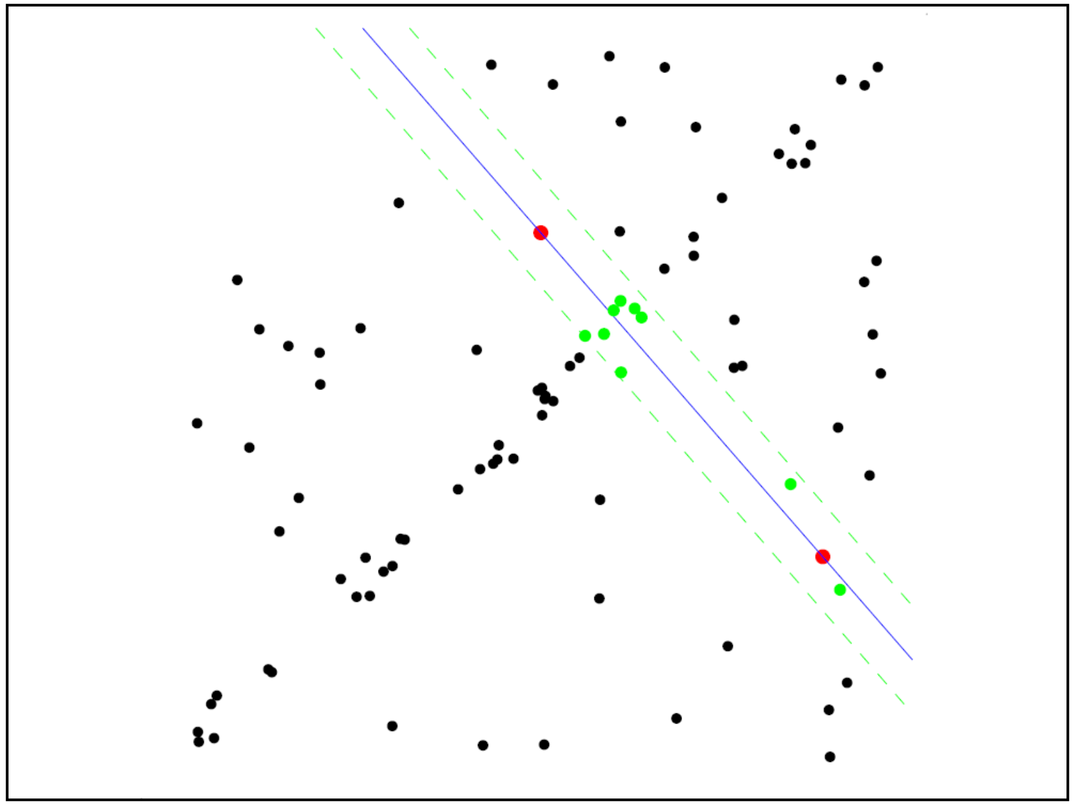
RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



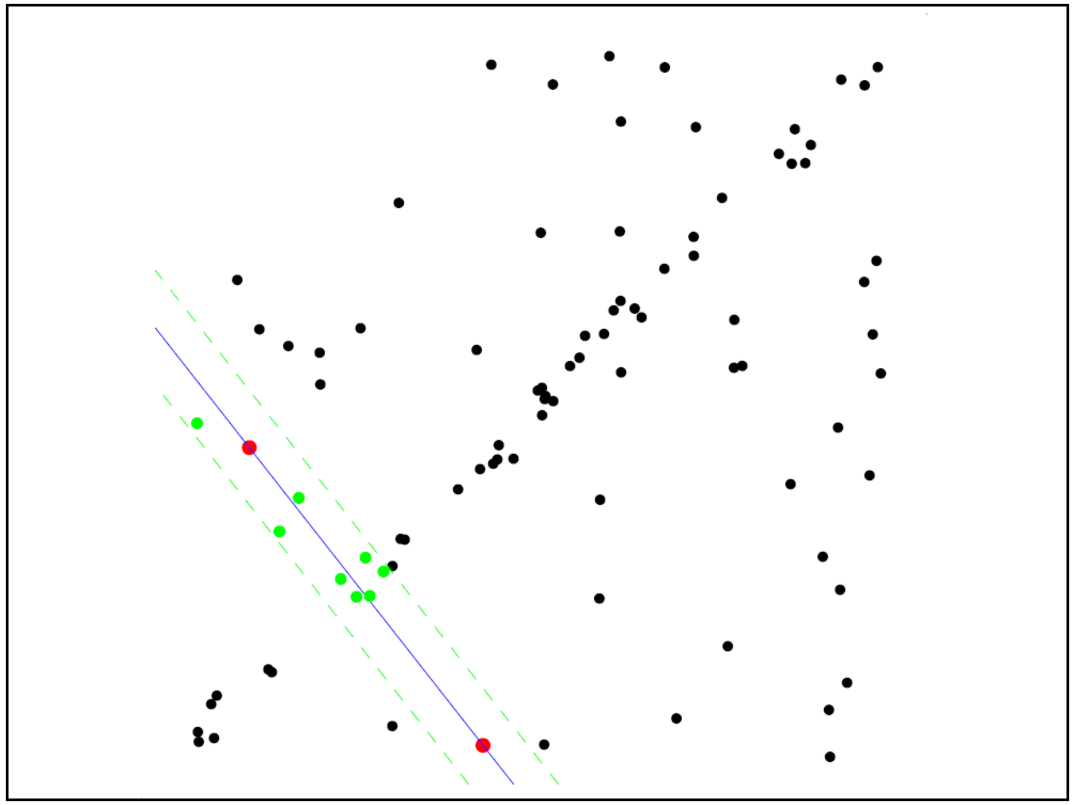
RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



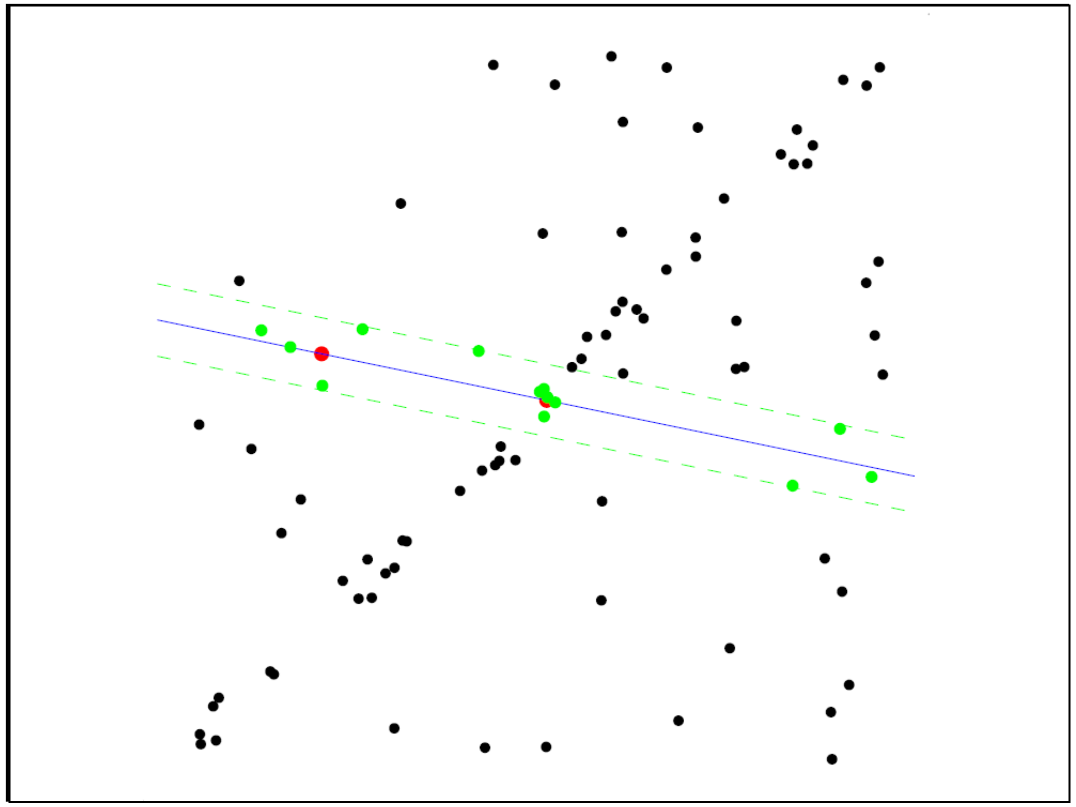
RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



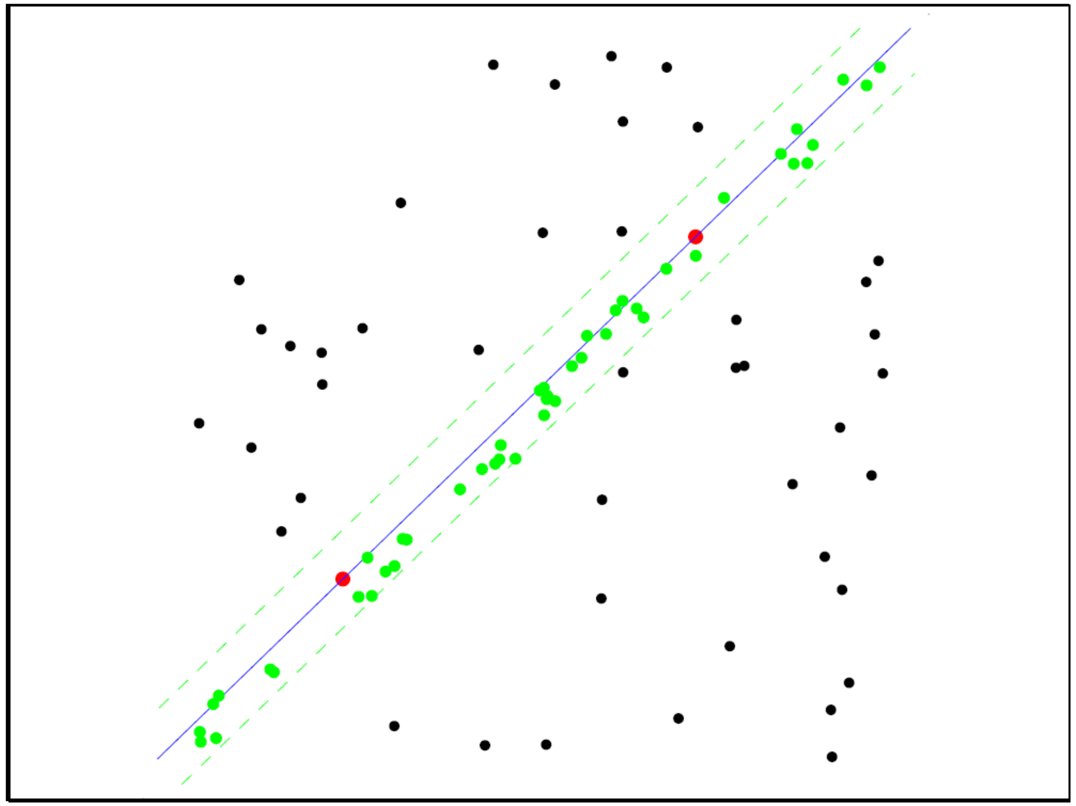
RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



RANSAC

```
Data: Set $S$ consisting of all $N$ points
Result: Set with maximum number of inliers
    (and corresponding fitting line)
while $i \leq k$ do
    randomly select 2 points from $S$;
    fit line $l_i$ through the 2 points;
    compute distance of all other points to line $l_i$ ;
    construct inlier set, i.e., count number of
        points with distance to the line less than $\gamma$;
    store line $l_i$ and associated set of inliers;
    $i \leftarrow i + 1$
end
Choose set with maximum number of inliers
```



RANSAC iterations

- In principle, one would need to check all possible combinations of 2 points in dataset
- If $|S| = N$, number of combinations is $\frac{N(N-1)}{2}$ → too many
- However, if we have a rough estimate of the percentage of inliers, we do not need to check all combinations…

RANSAC iterations: statistical characterization

- Let w be the percentage of inliers in the dataset, i.e.,

$$w = \frac{\text{number of inliers}}{N}$$

- Let p be the desired probability of finding a set of points free of outliers (typically, $p = 0.99$)
- Assumption: 2 points chosen for line estimation are selected independently
  - $P(\text{both points selected are inliers}) = w^2$
  - $P(\text{at least one of the selected points is an outlier}) = 1 - w^2$
  - $P(\text{RANSAC nevers selects two points that are both inliers}) = (1 - w^2)^k$

RANSAC iterations: statistical characterization

- Then minimum number of iterations $\bar{k}$ to find an outlier-free set with probability at least p is:

$$1 - p = (1 - w^2)^{\bar{k}} \Rightarrow \bar{k} = \frac{\log(1-p)}{\log(1-w^2)}$$

- Thus if we know w (at least approximately), after $\bar{k}$ iterations RANSAC will find a set free of outliers with probability p
- Note:
  - $\bar{k}$ depends only on w, not on N!
  - More advanced versions of RANSAC estimate w adaptively

Semantic information extraction

- Semantic information: *higher-level* scene information in sensor data (e.g., images) like objects, their locations, and relationships
- Encompasses a broad class of perception algorithms:
  - Object detection, semantic segmentation, object recognition, tracking
  - Conceptually: seeks to ground raw sensor data into structured information useful for downstream robot reasoning and action

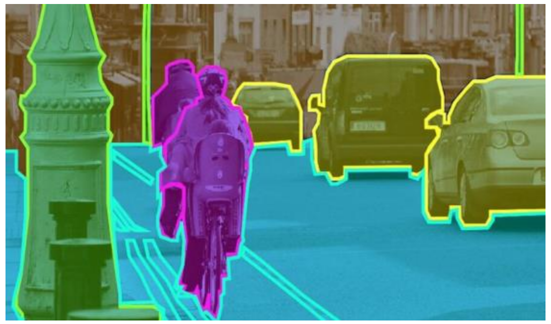
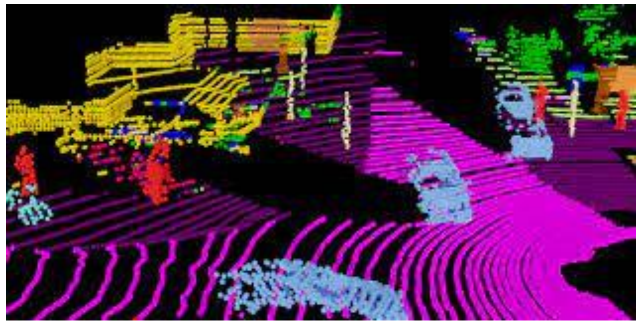


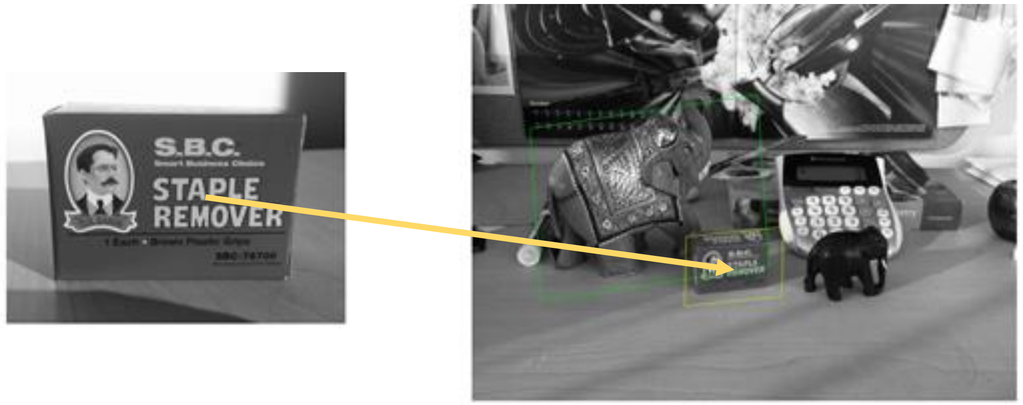
Image-based semantic segmentation



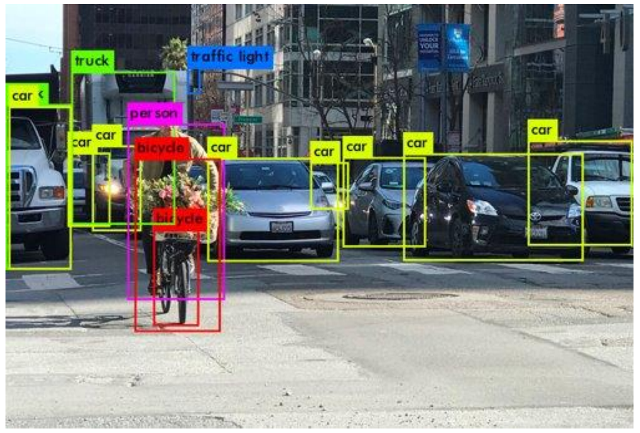
LiDAR-based semantic segmentation

Object detection

- Example of semantic extraction: object detection
  - Given a source image of an object, **localize the object** in the target image
  - What if the object is rotated, translated, scaled, partially occluded?
  - Solution: rely on stable feature detectors / descriptors for object detection



Today's detector
(feature-based, still relevant!)

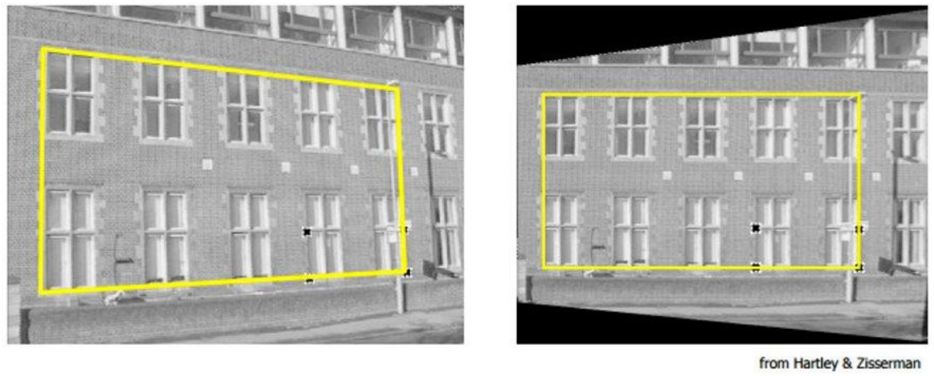


Modern detectors
(learned-based, DNNs)

Object detection

- The main problems in feature-based object detection are:
  a. Feature matching: detect and match object features across images
  b. Model fitting: fit *homography* to predict object location in the target image

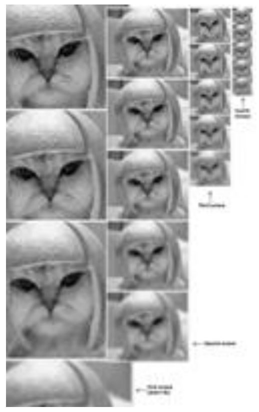
- Aside on homography
  - Maps plane in one image to plane in another image
  - Relevant for step "b." above



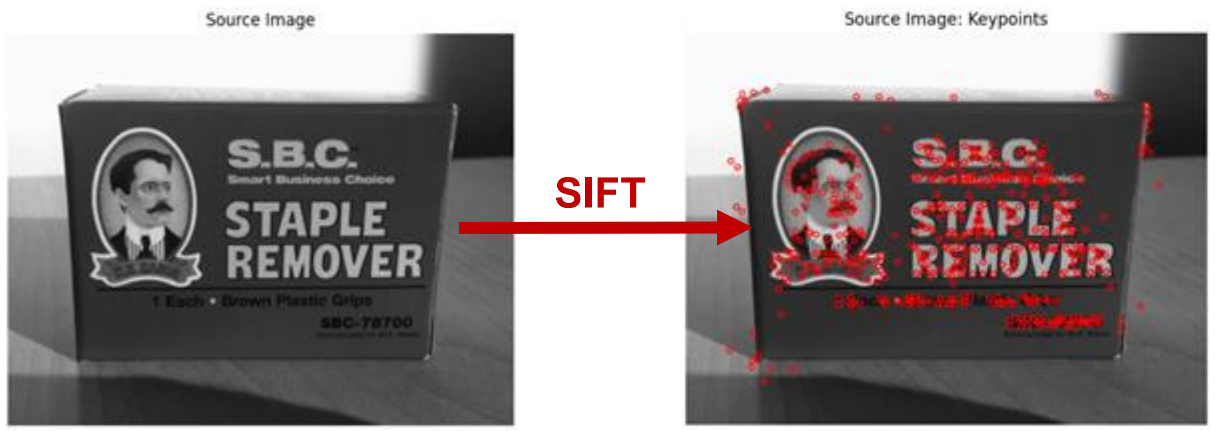
Projecting bounding box using homography

Step #1: Detect keypoints

- Goal: Detect *stable* and salient keypoints of the object
- Will make use of feature detectors and descriptors
  - Choices include SIFT, SURF, FAST, BRISK, ORB, amongst others
  - Many will work, some more efficiently or reliably depending on the setting
  - In this example, we use SIFT

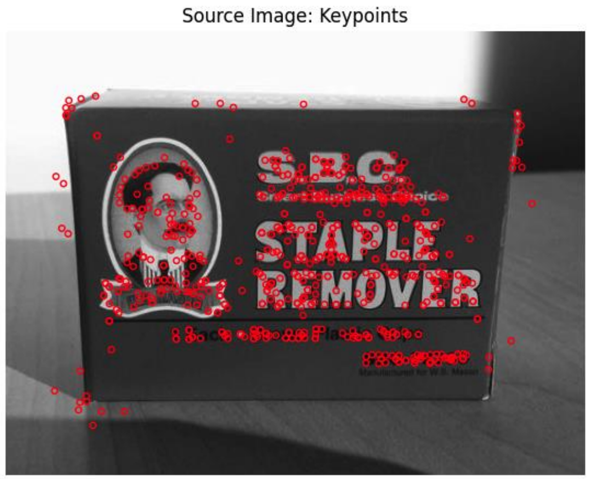


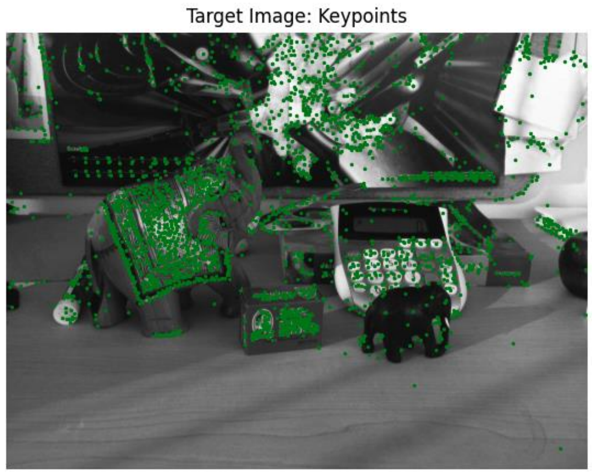
Scale invariance of SIFT



Step #1: Detect keypoints

- Goal: Detect *stable* and salient keypoints of the object
- Will make use of feature detectors and descriptors
  - Choices include SIFT, SURF, FAST, BRISK, ORB, amongst many others
  - Many will work, some more efficiently and/or reliably in a desired setting
  - In this example, we use SIFT





Q: But, how do we associate keypoints in the source image to keypoints in the target image?

Step #2: Match keypoints

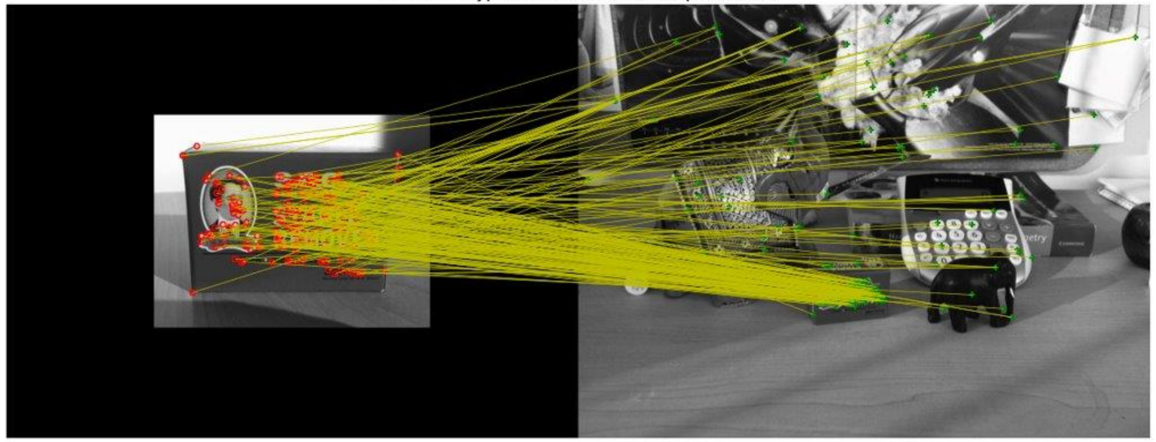
- Goal: Attempt to match keypoints across images
- Matching criterion depends on choice of descriptor
  - E.g., SIFT uses L2-norm, while ORB uses Hamming distance
  - Threshold match scores to get an <u>initial set</u> of correspondences

Careful, manually set "good" match thresholds

$$||f_{\text{SIFT}} - f'_{\text{SIFT}}|| < d_{max}$$

will often produce outliers!

Matched Keypoints: Unfiltered Correspondences



Step #3: Model fitting and outlier rejection

- Goal: Estimate homography between images and filter outliers
- Another application of RANSAC: fit the model (i.e., homography) while simultaneously rejecting outlier matches

Given hypothesis homography (H), a keypoint match

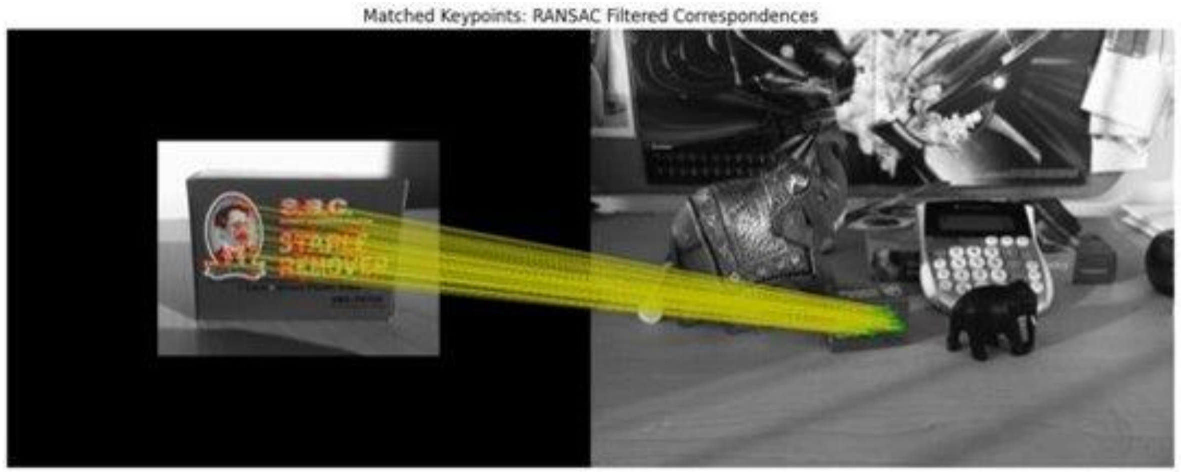
$$[\hat{u}', \hat{v}', 1]^T \propto p'_h = \mathbf{H} p_h = \mathbf{H}[u, v, 1]^T$$

is considered an "inlier" if

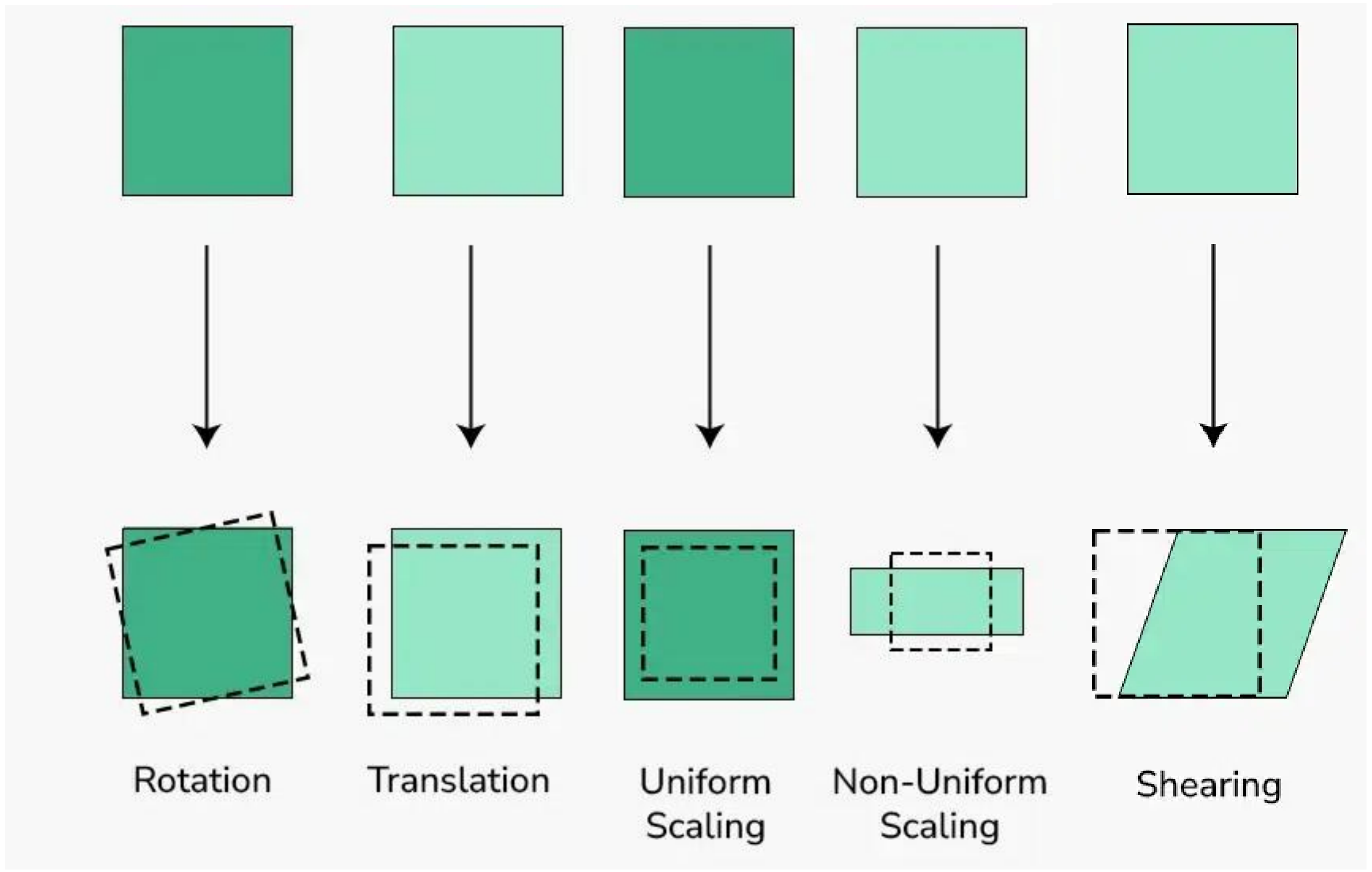
$$\sqrt{(u' - \hat{u}')^2 + (v' - \hat{v}')^2} < d_{\text{RANSAC}}$$

RANSAC in a nutshell:

1. **Find best homography H with the most inliers**
2. **Reject outliers under best homography H**



Recap Transform



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

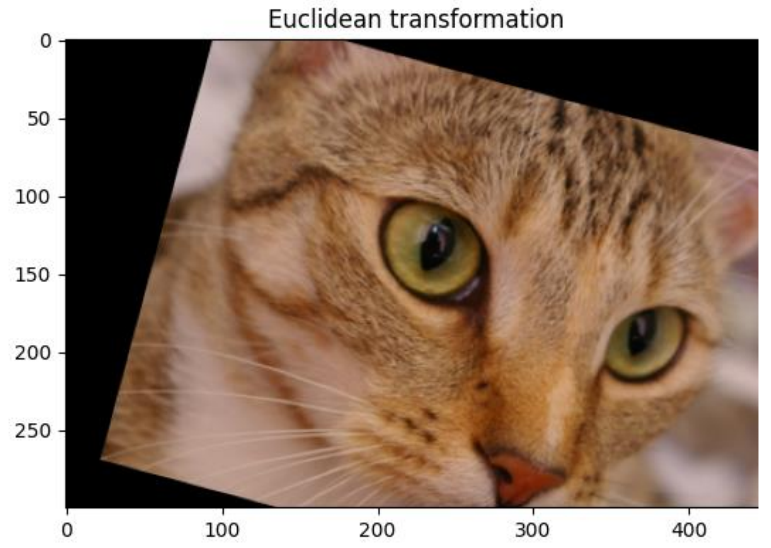
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

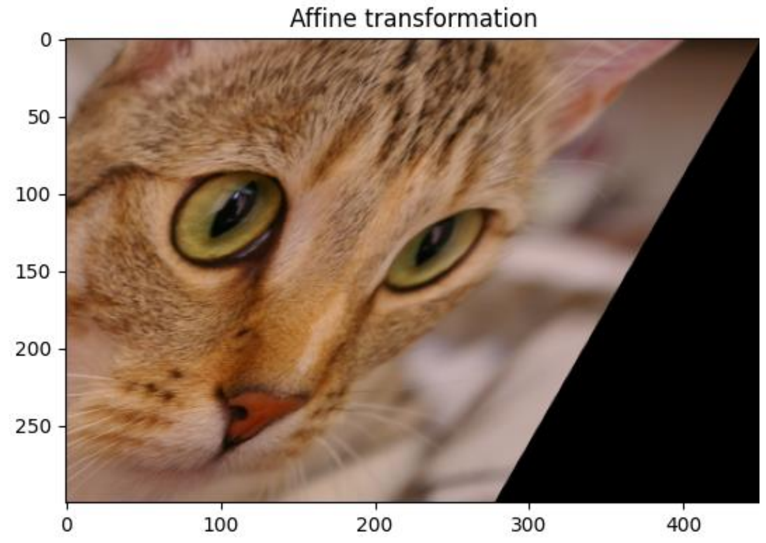
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k_y \\ k_x & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

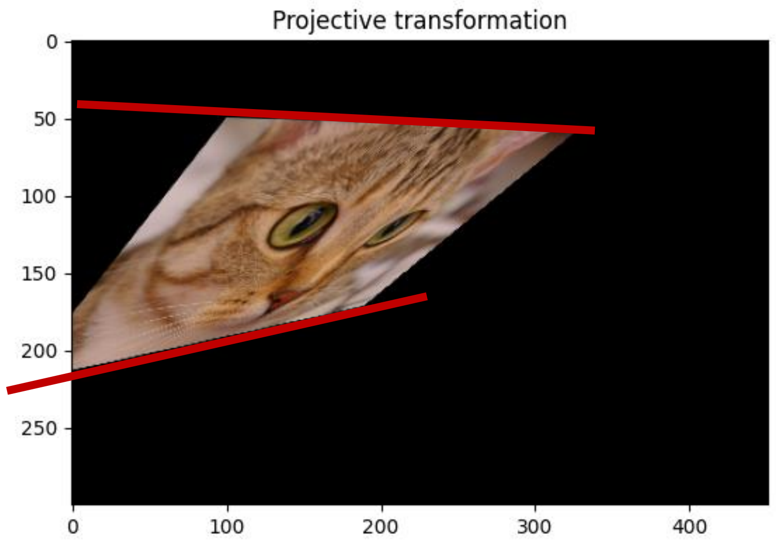
https://scikit-image.org/docs/0.24.x/auto_examples/transform/plot_transform_types.html

Recap Transform

Lines & Parallelism







$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = A \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

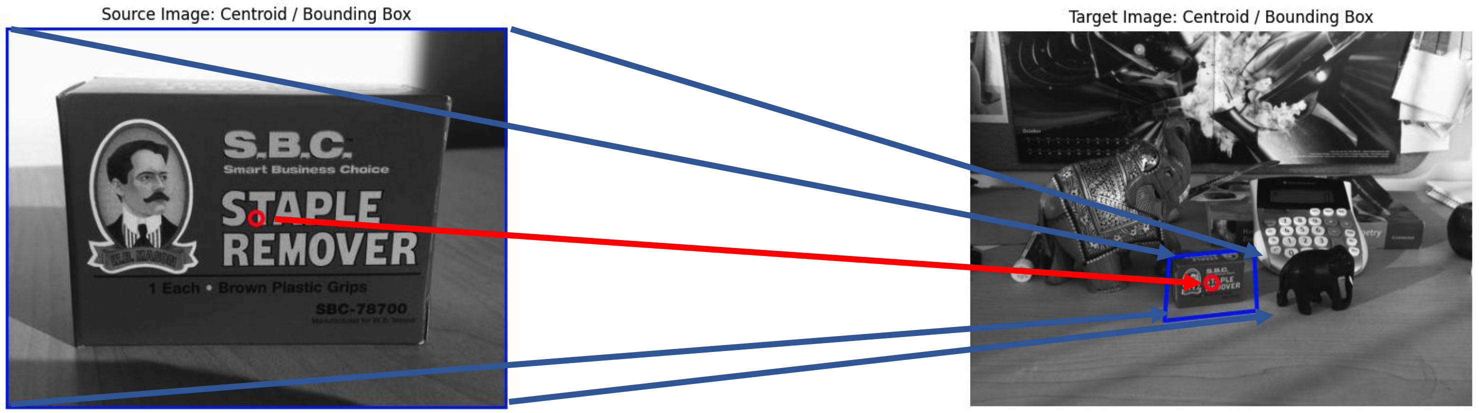
Affine Transform

$$\text{m} \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective / Perspective / Homography Transform

Step #4: Detect the object

- Goal: Use homography (H) to localize object in target image
  - Simply project object centroid and/or bounding box corners from source image to target image
  - Note: Homographies are expressive but do not maintain parallelism – we may not get a bounding "box" in the target image! Other transformations (e.g., affine), are possible too

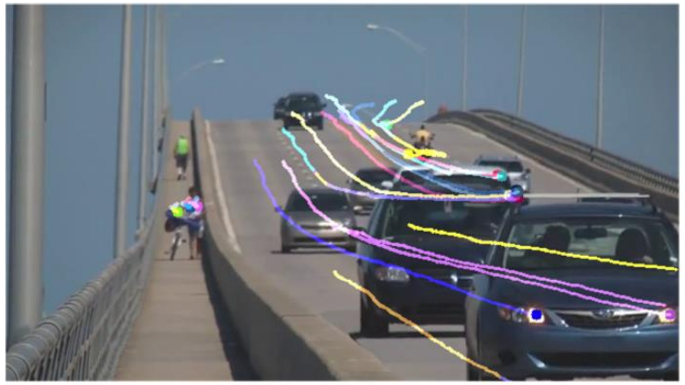

Source Image: Centroid / Bounding Box
Target Image: Centroid / Bounding Box

$$p_h = [u, v, 1]^T$$

$$[\hat{u}', \hat{v}', 1]^T \propto p_h' = \mathbf{H} p_h = \mathbf{H}[u, v, 1]^T$$

Object tracking

- Once objects are detected, how can we track them over time?
  - Re-running object detection from scratch at each frame can be slow!
  - Instead, object tracking *exploits existing knowledge* of the object (e.g., detected position) to track its motion over a sequence of images
  - The problem is equivalent to estimating pixel velocities (optical flow)

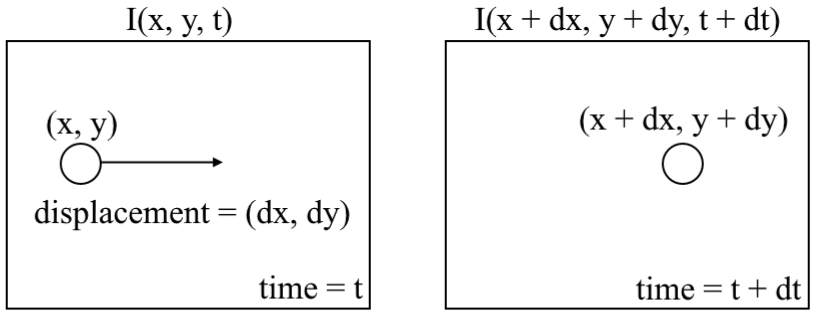


Sparse optical flow (tracking keypoints)

Dense optical flow (tracking all pixels)

Object tracking

- **Intuition:** pixel motion is small across frames
  - Assumption: only need to search within a local region



source

- We can express the optical flow problem as:

$$I(x,y,t) = I(x+\delta x, y+\delta y, t+\delta t)$$

(Taylor expansion step) $$\approx I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$$

Optical flow equation*

$$\implies \frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} = 0$$

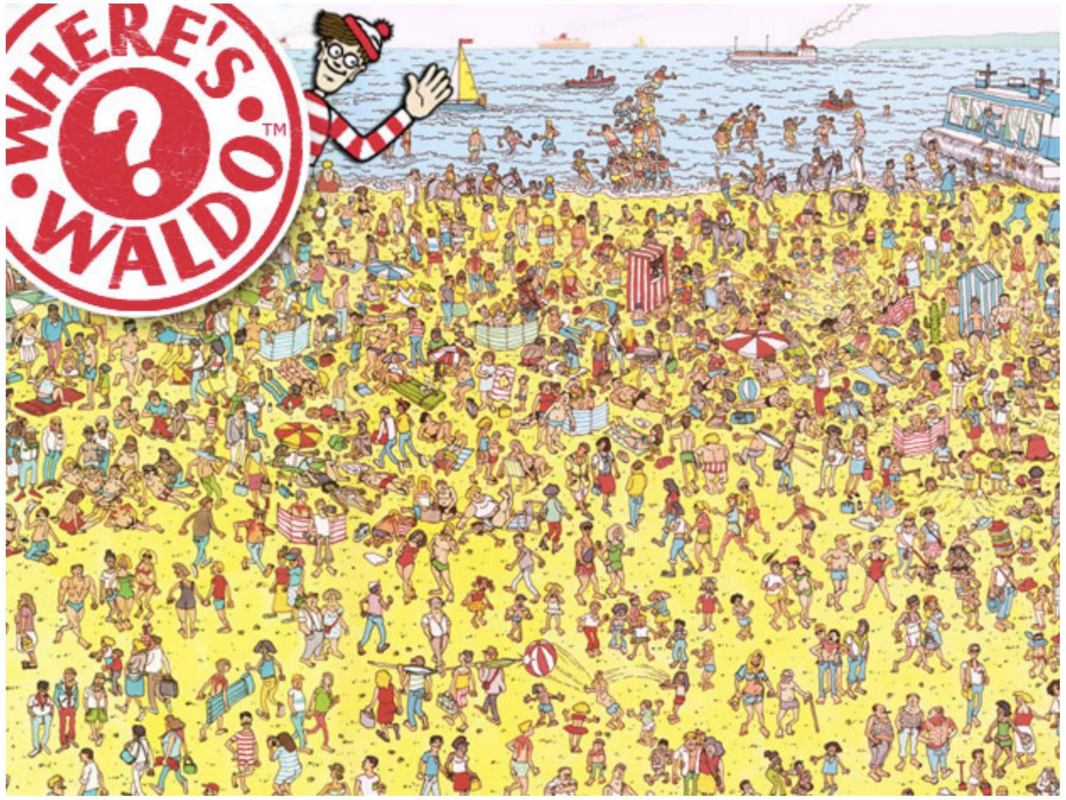
- Solving the optical flow equation gives pixel velocities v_x , v_y
  - Many sparse and dense optical flow techniques have been developed, for example, the Lucas-Kanade method (sparse) and the Gunnar-Farneback method (dense)

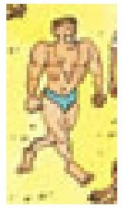
Object recognition

- Object recognition: capability of naming discrete objects in the world
- Why is it hard? Many reasons, including:
    1. Real world is made of a jumble of objects, which all occlude one another and appear in different poses
    2. There is a lot of variability intrinsic within each class (e.g., dogs)
- In this class, we will look at two methods:
    1. Template matching (classic)
    2. Neural network methods (treated as a black box, see next lecture)

Template matching

- How can we find this guy?





Source: Sanja Fidler

Template matching

- Slide and compare!

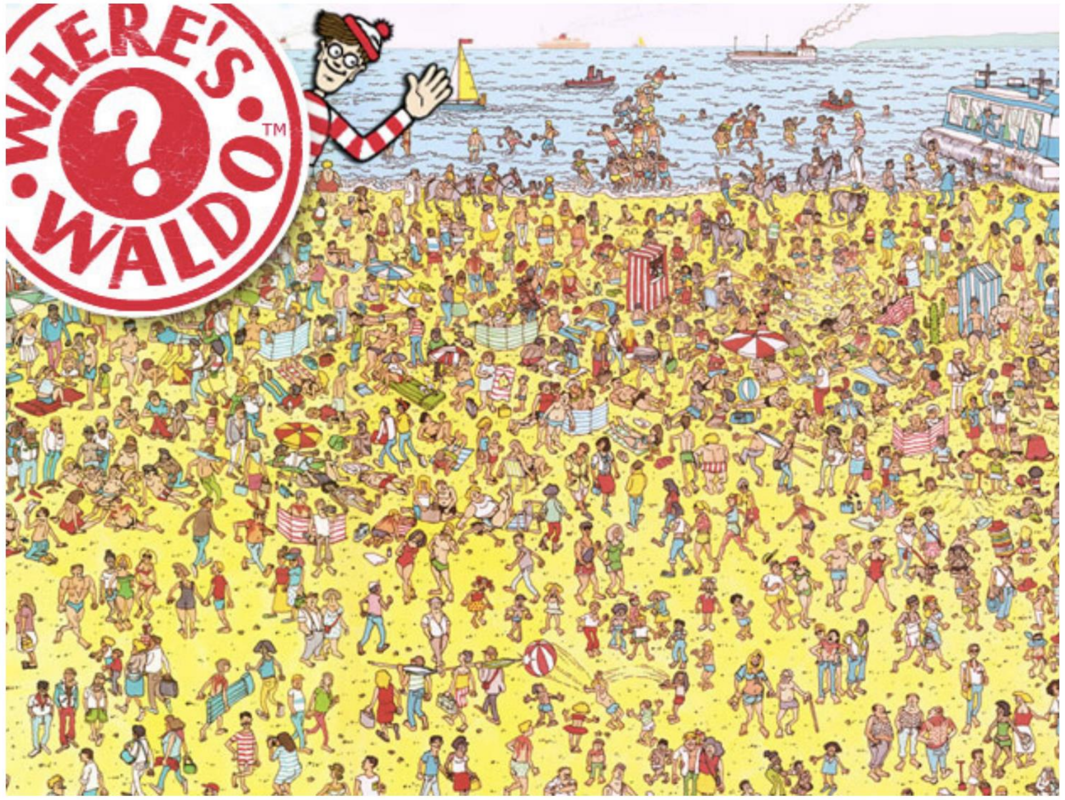


Image I

Filter F

Source: Sanja Fidler

Template matching

- In practice, remember correlation:

$$I'(x,y) = F \circ I = \sum_{i=-N}^{N} \sum_{j=-M}^{M} F(i,j) I(x+i, y+j)$$

- One can equivalently write: $I'(x,y) = \underbrace{\overset{\text{Vector representation of filter}}{\mathbf{f}^{\mathrm{T}}} \cdot \mathbf{t}_{\mathrm{ij}}}$

Vector representation of filter

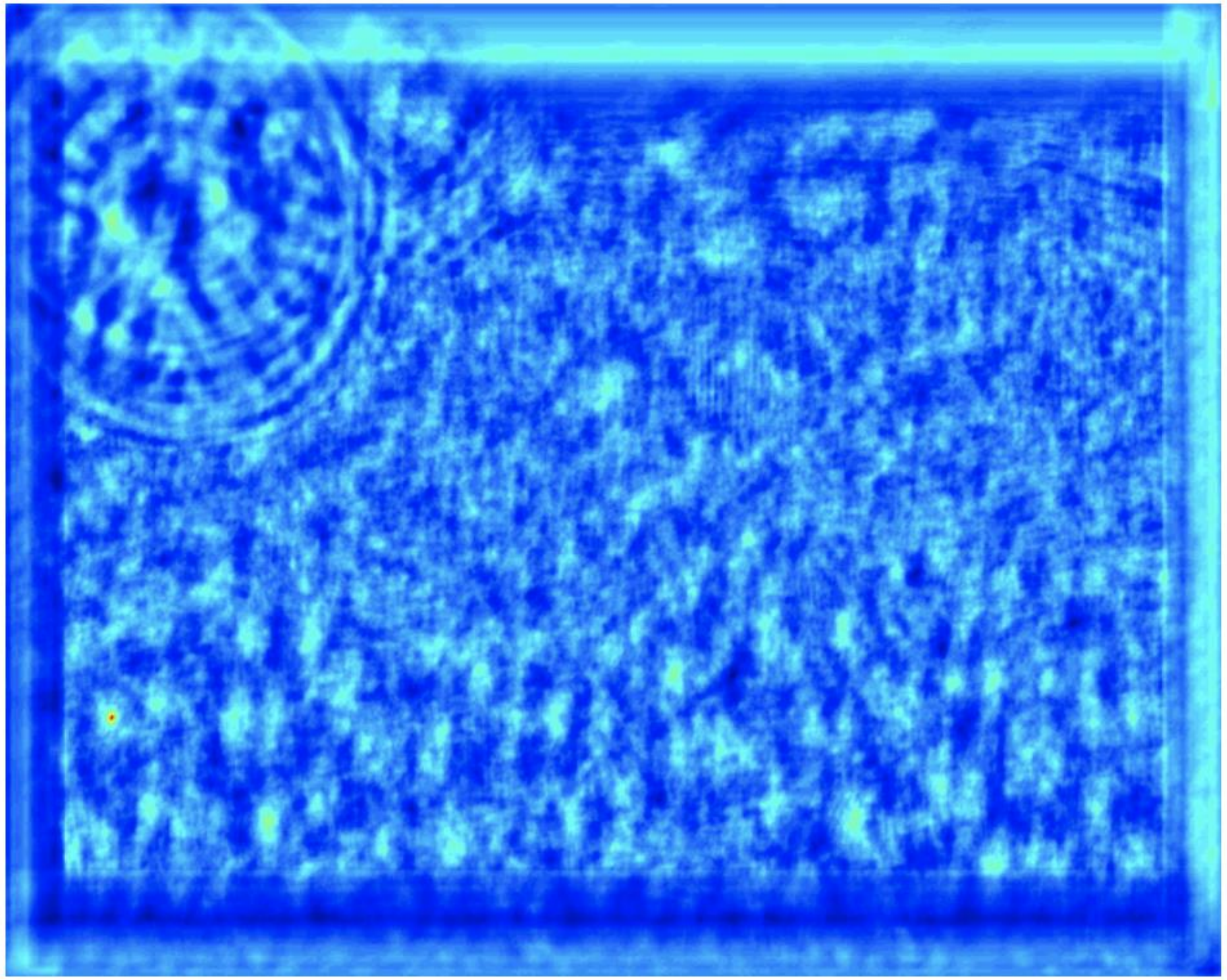
Vector representation of neighborhood patch

- To ensure that perfect matching yields one, we consider *normalized* correlation, that is

$$I'(x,y) = \frac{\mathbf{f}^{\mathrm{T}} \cdot \mathbf{t}_{\mathrm{ij}}}{||\mathbf{f}|| \, ||\mathbf{t}_{\mathrm{ij}}||}$$

Template matching

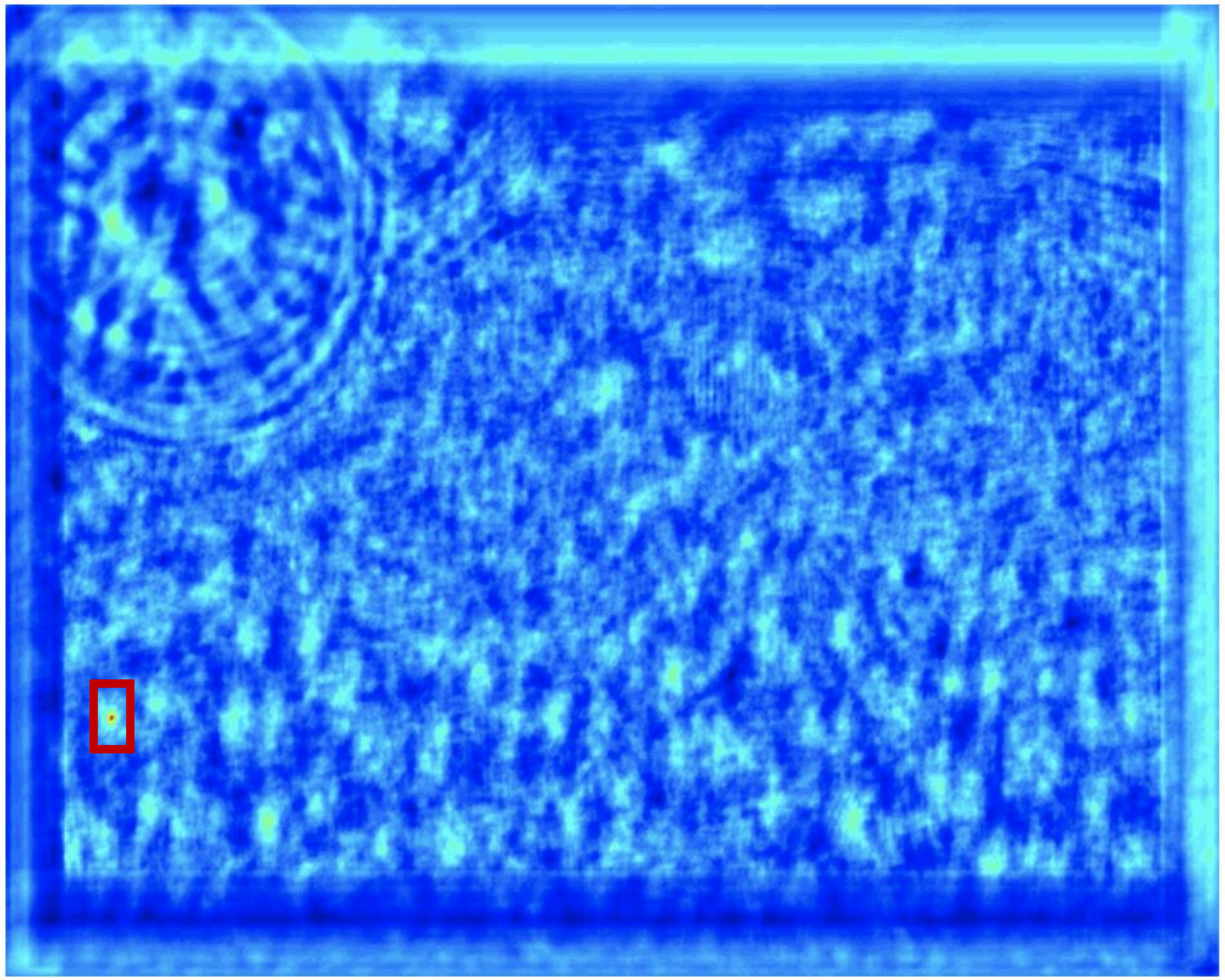
Result:



Source: Sanja Fidler

Template matching

Result:



Source: Sanja Fidler

Template matching

- Problem: what if the object in the image is much larger or much smaller than our template?
- Solution: re-scale the image multiple times, and do correlation on every size!
- This leads to the idea of *image pyramids*

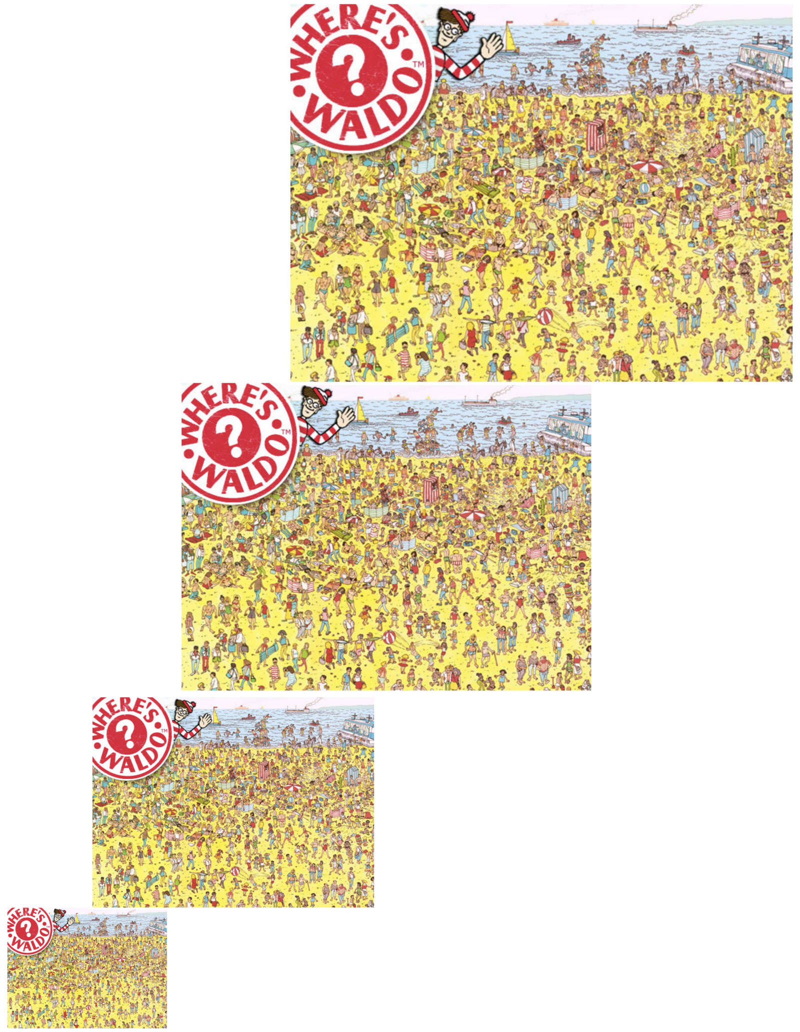
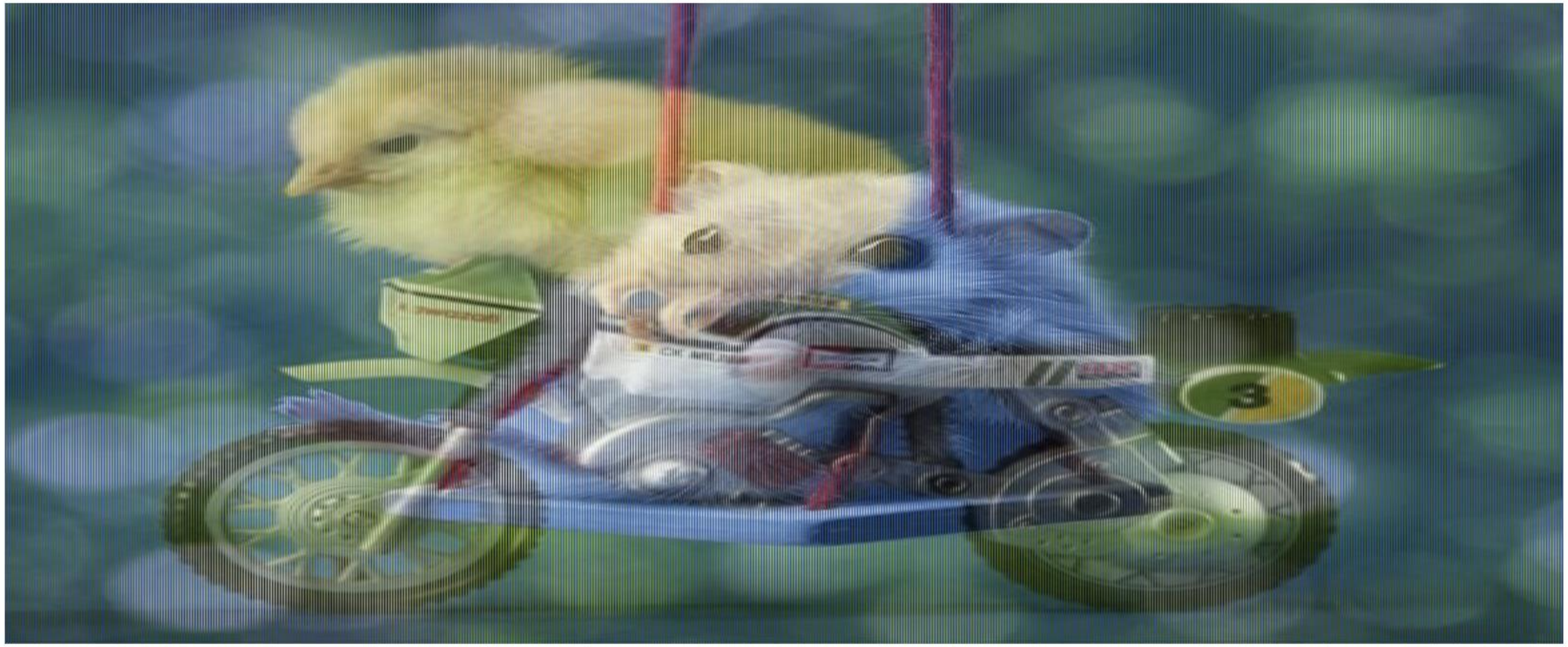


Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction



Source:
Sanja Fidler

Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction

Source:
Sanja Fidler

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image

Source:
Sanja Fidler

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image

Source: Sanja Fidler

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image

Source:
Sanja Fidler

Image pyramids

- A sequence of images created with Gaussian blurring and down-sampling is called a Gaussian pyramid
- The other step is to perform up-sampling (nearest neighbor, bilinear, bicubic, etc.)

However, classical methods can be brittle!

- Sensitive to variations in rotation, etc.
- Loss of spatial information
- Lack of robustness (to partial occlusions, deformations, etc.)

Using learned features

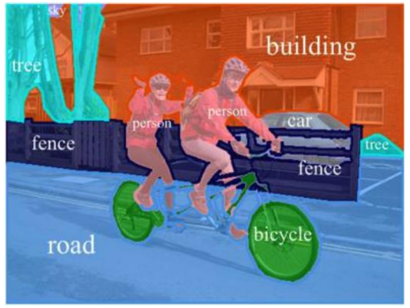
Solution: Use learned features!

- We can use convolutional neural networks (CNNs) to detect and describe features
- Convolutional neural networks (CNNs): deep learning models for processing structured grid data, such as images, by using layers of convolutional operations to automatically learn hierarchical features and patterns

Uses in modern computer vision

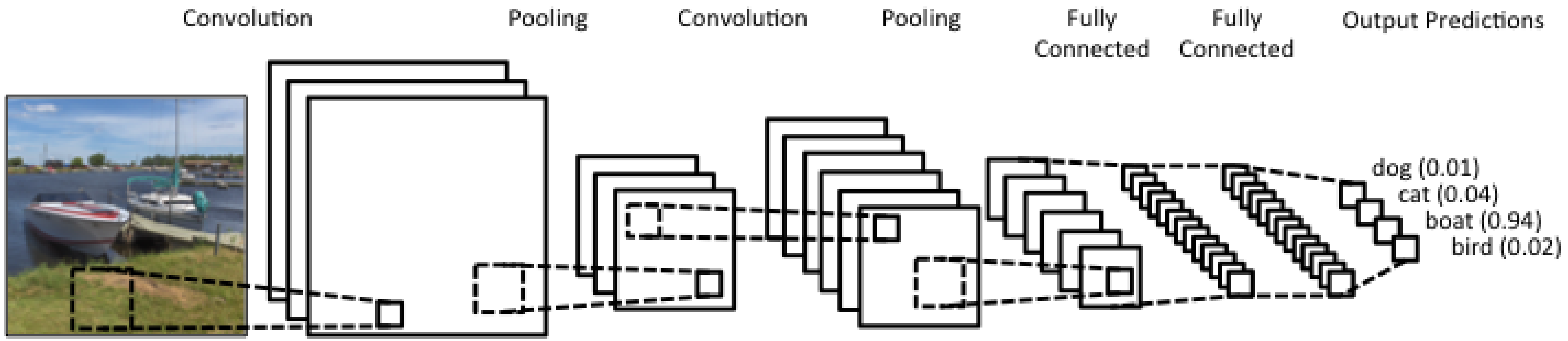
- Using CNNs for computer vision tasks took off ~2012 with the success of the AlexNet architecture for image classification on the ImageNet dataset
- Today, learned features are used in many applications: image classification, object detection, image segmentation, object tracking, image generation etc.
- Modern models also include GANs, transformers, etc.

Classification: Goldfish



Semantic segmentation

Next time


Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)