LECTURE 1

Geometry 3D and images:

PRESERVE: - straight lines are mapped to straight lines
- size of objects is inverse proportional to the distance

NOT PRESERVE: - angles
- parallels lines meet at the vanishing point of their direction.

Trasformations in projective geometry

- translation $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$ #dof: 2

- euclidian $\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$ #dof: 3

- similarity $\begin{bmatrix} s(\cos\theta) & s(-\sin\theta) & t_x \\ s(\sin\theta) & s(\cos\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix}$ #dof: 4

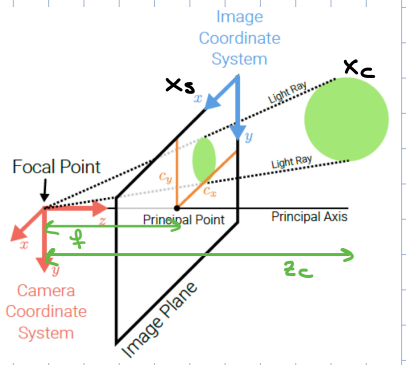
- affine (+shearing) $\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix}$ #dof: 6

- projective $\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$ #dof: 8 (up to scale)

Converting a point in the 3D space in the Camera Coordinate System in the image plane

$$\vec{x}_s = \underbrace{\begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{calibration matrix } K} \vec{x}_c$$

(skew; principal point offset)

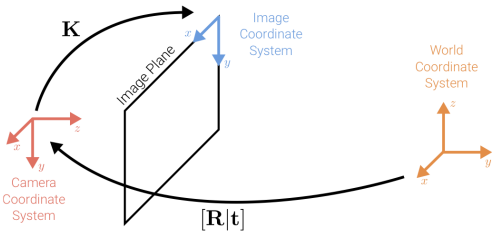


- calibration matrix contains the camera intrinsic parameters.
- skew is needed if the sensor is not mounted perpendicular to the optical axis.
- usually $f_x = f_y$ and $s = 0$

Chaining transformations

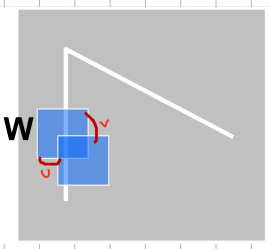
$$\vec{x}_s = \underbrace{\begin{bmatrix} K & 0 \end{bmatrix}}_{\substack{\text{camera intrinsic parameters} \\ (3\times 4)}} \underbrace{\begin{bmatrix} R & t \\ 0^t & 1 \end{bmatrix}}_{\substack{\text{camera pose} \\ (4\times 4)}} \vec{x}_w = K[R\ t]\,\vec{x}_w = P\,\vec{x}_w$$

#dof = 5+6 = 11



LECTURE 2

Keypoint Localization - Harris Detector



Comparing each pixel before and after by summing up the squared differences (SSD):

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

If the motion (u,v) is small, then the 1st order approx. is good:

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v = I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) = \sum_{(x,y)\in W} \left[\begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} \right]^2 \qquad E(u,v) = \sum_{(x,y)\in W} \begin{bmatrix} u & v \end{bmatrix}\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

second order moment matrix

$$M$$

NO ROTATION INVARIANT

$$\sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

ROTATION INVARIANT (smooth with gaussian)

$$\sum_{(x,y)} g(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

gaussian filter over the window

Thresholding eigenvalues to detect corners

- Harris & Stephens : $R = \det(M) - k\,\text{trace}^2(M) = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2$
- Kanade & Tomasi : $R = \min(\lambda_1, \lambda_2)$

Recap: Harris detector

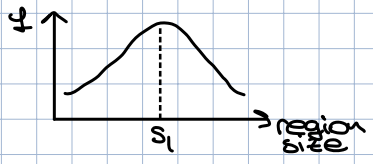
1. compute the M matrix
2. compute the cornerness function R
3. perform non-maximum suppression

(if 2 pixels are close to each others, there are not two corners: it's the same one. Non-maximum suppression is an heuristic approach where you pick the maximum response in a local region).

In order to compare points, we need to compute a descriptor over a region that must be scale-invariant.

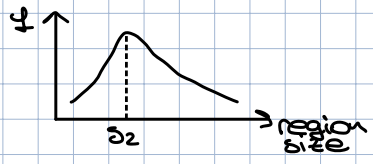
Automatic Scale Selection

1. Design a signature function on the region that is "scale invariant". For a point in one image, we can consider a function of a region size.
2. Take the local maximum of this function.



$s_2 = 1/2 \cdot s_1$

SCALE = 1/2



3. Normalize: rescale to fixed size.

Signature functions:

- Laplacian-of-Gaussian = "blob" detector $\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$
  characteristic scale: scale that produces peak in Laplacian response.

- Difference-of-Gaussian: fast approximation of Laplacian-of-Gaussian

Local descriptors

Descriptor	Features
Image patch	Only if geometry and appearance is unchanged
Image gradients	Invariant to absolute intensity values
Color histogram	Invariant to changes in scale and rotation
Spatial histogram	Some invariance to deformations

Orientation normalization

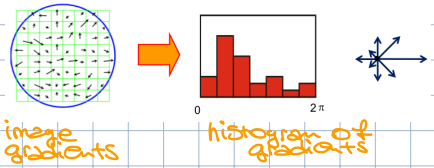


image gradients — histogram of gradients

The dominant gradient is found and the patches are rotated to the dominant direction.

SIFT - Scale Invariant Feature Transform

Algorithm:
1. Scale-space extrema detection
2. Keypoint localization
3. Keypoint descriptor
   - divide patch into 4x4 sub-patches: 16 cells
   - compute histogram of gradient orientations (8 reference angles)
   - resulting descriptor: 4x4x8 = 128 dimentions
4. orientation normalization
6. Keypoint matching

LECTURE 3

Optical flow

Motion field = 2D motion field representing the projection of the 3D motion of points in the scene onto the image plane.

Optical flow = 2D velocity field describing the apparent motion in the images.

Given 2 consecutive image frames, estimate the motion of each pixel.

Assumptions:

- **brightness constancy** (color constancy) allows for pixel comparison

$$I(x(t), y(t), t) = C_{constant}$$

- **small motion** linearization of the brightness constancy constraint

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

BRIGHTNESS CONSTANCY EQUATION

$$\frac{\partial I}{\partial x}\cdot\frac{\partial x}{\partial t} + \frac{\partial I}{\partial y}\cdot\frac{\partial y}{\partial t} + \frac{\partial I}{\partial t} = 0$$

$$I(x(t), y(t), t) \approx I(x(t), y(t), t) + \frac{\partial I}{\partial x}\partial x + \frac{\partial I}{\partial y}\partial y + \frac{\partial I}{\partial t}\partial t \qquad \text{taylor expansion}$$

$$\frac{\partial I}{\partial x}\partial x + \frac{\partial I}{\partial y}\partial y + \frac{\partial I}{\partial t}\partial t = 0 \qquad /:\partial t$$

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

$$I_x \cdot u + I_y \cdot v + I_t = 0 \qquad \text{shorthand notation}$$

$$\nabla I^T v + I_t = 0 \qquad \text{vector form}$$

How to compute:

- $I_x = \frac{\partial I}{\partial x}$; $I_y = \frac{\partial I}{\partial y}$ → Sobel filter / Derivative-of-Gaussian filter
- $I_t = \frac{\partial I}{\partial t}$ → frame differencing ($\forall$ pixel $I_t = I(t) - I(t+1)$)
- $u = \frac{\partial x}{\partial t}$; $v = \frac{\partial y}{\partial t}$ → unknown in the optical flow problem

Where do we get more equations?

- **Lucas-Kanade flow**: "constant" flow → flow is constant for all pixels (local method - sparse)

Assume that the surrounding patch (say 5x5) has constant flow.
Using 5x5 image patch, gives 25 equations.

$$I_x(p_i)u + I_y(p_i)v = -I_t(p_i) \qquad i = 1, \dots, 25$$

equivalent in solving: $A^TA \cdot \hat{x} = A^Tb$

$$\hat{x} = (A^TA)^{-1}A^Tb$$

$$\underbrace{\begin{bmatrix} \sum_{p\in P} I_xI_x & \sum_{p\in P} I_xI_y \\ \sum_{p\in P} I_yI_x & \sum_{p\in P} I_yI_y \end{bmatrix}}_{A^TA} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\hat{x}} = -\underbrace{\begin{bmatrix} \sum_{p\in P} I_xI_t \\ \sum_{p\in P} I_yI_t \end{bmatrix}}_{A^Tb}$$

where summation is over each pixel p in patch P

Implications:
- corners are when λ_1, λ_2 are big.
- corners are regions with two different gradient dir.
- corners are good places to compute flow

- **Horn-Schunck flow**: "smooth" flow → flow can vary from pixel to pixel (global method - dense)

$$\min_{(u,v)} \sum_{i,j} \left\{ E_s(i,j) + \lambda E_d(i,j) \right\}$$

$$E_s = \frac{1}{4}\left[(u_{ij} - u_{i+1,j})^2 + (u_{ij} - u_{i,j+1})^2 + (v_{ij} - v_{i+1,j})^2 + (v_{ij} - v_{i,j+1})^2 \right]$$

smoothness constancy

$$E_d = \left[I_x u_{ij} + I_y v_{ij} + I_t \right]^2$$

brightness constancy

$$\hat{u}_{kl} = \bar{u}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x \qquad \hat{v}_{kl} = \bar{v}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$$

Algorithm:

1. precompute I_y, I_x, I_t
2. Initialize flow field $\begin{cases} u = 0 \\ v = 0 \end{cases}$
3. while NOT converged compute flow field updates for each pixel

$$\hat{u}_{kl} = \bar{u}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x \qquad \hat{v}_{kl} = \bar{v}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$$

$\bar{u}_{kl}$ = old average
$\hat{u}_{kl}$ = new value

LECTURE 4

Machine learning basics

Linear regression

Goal: predict y for a previously unseen input x

Error function:

$$f(x,w) = w^T x \qquad x = (1,x)^T$$

$$E(w) = \sum_i (f(x_i,w) - y_i)^2$$

$$= \|Xw - y\|_2^2$$

$$\nabla_w E(w) = \nabla_w (Xw-y)^T(Xw-y) = 2X^TXw - 2X^Ty$$

$$\stackrel{!}{=} 0 \implies \hat{w} = \underbrace{(X^TX)^{-1}X^T}_{\text{Moore-Penrose inverse}} y$$

Polynomial curve fitting

Same as linear regression but $x = (1, x^1, x^2, \dots, x^M)^T$

$$X = \begin{pmatrix} \vdots & \vdots & \vdots & & \vdots \\ 1 & x_i & x_i^2 & \dots & x_i^M \\ \vdots & \vdots & \vdots & & \vdots \end{pmatrix} \qquad y = \begin{pmatrix} \vdots \\ y_i \\ \vdots \end{pmatrix} \qquad w = \begin{pmatrix} w_0 \\ \vdots \\ w_m \end{pmatrix}$$

CAPACITY: complex of fundions which can be represented by model f $(=M)$
UNDERFITTING: low capacity
OVERFITTING: nigh capacity

General approach: 1. choose hyperparameter $(M, \eta, \dots)$ and train model 60%
2. k-fold cross validation on test-set 20%
3. validation set to confirm 20%

Ridge regression

$$E(w) = \|Xw - y\|_2^2 + \lambda \|w\|_2^2$$

$$\nabla_w E(w) = 2X^TXw - 2X^Ty + 2\lambda w$$

$$\stackrel{!}{=} 0 \implies \hat{w} = (X^TX + \lambda I)^{-1} X^T y$$

Maximum Likelihood Estimation

$$\hat{w}_{ML} = \operatorname{argmax}_w p(y|X,w) = \operatorname{argmax}_w \prod_i p(y|X,w)$$

$$= \operatorname{argmax}_w \sum_i \log p(y_i|x_i,w) \qquad \text{assuming } p(y_i|x_i,w) \sim \mathcal{N}(w^T x_i, \sigma)$$

$$= \underbrace{\operatorname{argmin}_w \|Xw-y\|_2^2}_{\text{gaussian distribution}} \quad f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \qquad \underbrace{\operatorname{argmin}_w \|Xw-y\|_1}_{\text{Laplace distribution}} \quad f(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

Logistic regression

$p(y|x,w)$ = Bernoulli distribution

$= \theta^y (1-\theta)^{(1-y)}$

$\theta = f_w(x)$ – sigmoid function $\sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$

$$\hat{w}_{ML} = \arg\max \sum_i \log p(y_i|x_i;w)$$

$$= \arg\max \sum_i \log \left[\theta_i^{y_i} (1-\theta_i)^{(1-y_i)} \right]$$

$$= \arg\min \sum_i \boxed{-y_i \log \theta_i - (1-y_i) \log(1-\theta_i)}$$

binary cross entropy loss $\mathcal{L}(\theta_i, y_i)$

→ NOT quadratic ⇒ gradient descent

Gradient Descent

Algorithm:

1. Initialize weights w^0 and pick learning rate η
2. For all data points $i \in \{1, \dots, N\}$ do:
   - 2.1 Forward propagate x_i through the network to calculate prediction $\hat{y}_i$
   - 2.2 Backpropagate to obtain gradient $\nabla_w \mathcal{L}_i(w^t) \equiv \nabla_w \mathcal{L}(\theta_i, y_i, w^t)$
3. Update gradients: $w^{t+1} = w^t - \eta \frac{1}{N} \sum_i \nabla_w \mathcal{L}_i(w^t)$

Expensive since: $\begin{cases} \dim(w) \geq 1 \text{ million} \\ N \geq 1 \text{ million} \end{cases}$

Stochastic Gradient Descent

Algorithm:

1. Initialize weights w^0 and pick learning rate η and minibatch size $|x_{batch}|$
2. Draw random minibatch $\{(x_1, y_1), \dots, (x_B, y_B)\} \subseteq X$ (with $B \ll N$)
3. For all minibatches elements be $\{1, \dots, B\}$ do:
   - 3.1 Forward propagate x_i through the network to calculate prediction $\hat{y}_b$
   - 3.2 Backpropagate to obtain gradient $\nabla_w \mathcal{L}_b(w^t) \equiv \nabla_w \mathcal{L}(\theta_b, y_b, w^t)$
4. Update gradients: $w^{t+1} = w^t - \eta \frac{1}{B} \sum_b \nabla_w \mathcal{L}_b(w^t)$
5. If validation error decreases, go to step 2, otherwise stop.

Deep neural network basics

Computation Graphs - Backpropagation

- we can write mathematical expressions as a computation graph
- values are efficiently computed forward, gradients backwards
- multiple incoming gradients are summed up (multivariate chain rule)
- modularity: each node must only "know" how to compute gradients w.r.t. its own arguments

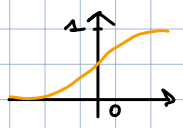
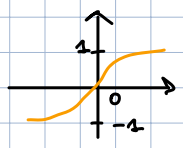
Bauer's Formula: let G be a computation graph. For any two nodes i and j in G, let $P(i,j)$ be the set of all directed paths starting at j and ending at i. The partial derivative of z_i with respect to z_j is then equal to

$$\frac{\partial z_i}{\partial z_j} = \sum_{\vec{\pi} \in P(i,j)} \prod_{(u,e) \in \vec{\pi}} \frac{\partial z_e}{\partial z_u}$$

Multi-Layer Perceptrons

Each layer i comprises multiple neurons j which are implemented as affine transformations ($a^T x + b$) followed by non-linear activation functions (g).

$$h_{ij} = g(a_{ij}^T h_{i-1} + b_{ij})$$

- Activation functions $g(\cdot)$
  - Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$  Smooth vanishing
  - Tanh $\tanh(x)$  Smooth vanishing
  - ReLU $\max(0,x)$  quick to compute the gradient vanishing
  - LeakyReLU $\max(0.1x, x)$  quick to compute the gradient
  - Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$
  - ELU $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

With linear activation function (ex. $g(x) = x$), a multi-layer network can only express linear functions.

Universal approximation theorem

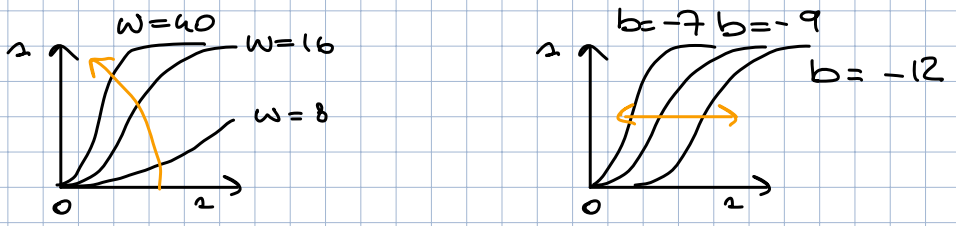
Let σ be any continuous discriminatory function. Then finite sums of the form

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(a_j^T x + b_j)$$

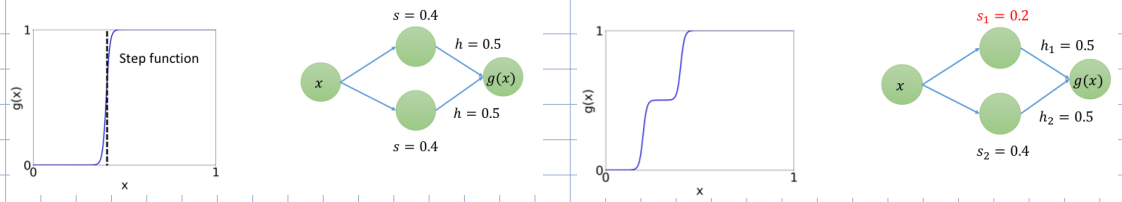
are dense in the space of continuous functions $C(I_n)$ on the n-dimensional unit cube I_n. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$ for which

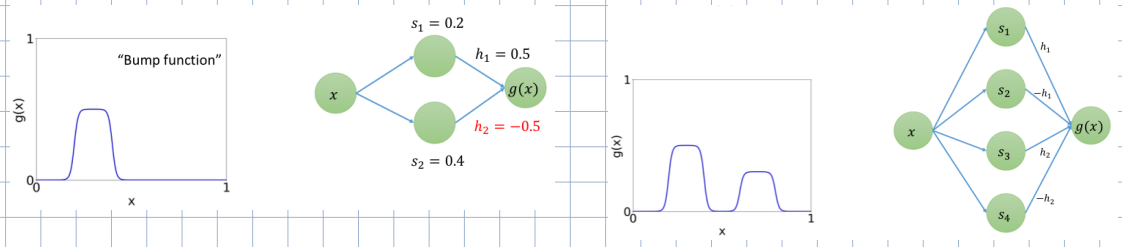
$$|G(x) - f(x)| < \varepsilon \quad \text{for all } x \in I_n$$

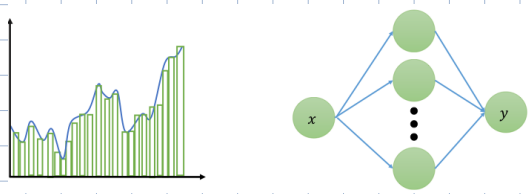
- SIGMOID FUNCTION shape: $g(x) = \sigma(w^T x + b)$



Learning edge $s = -\frac{b}{w} = 0,4$







The theorem says that feed forward networks can approximate $f(x)$. It does not ten anything about the chances of learning the correct parameters.

Regression problem

penalizes outliers more

Gaussian distribution ∩ thin tail

$$p(y|x,w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - f_w(x))^2}{2\sigma^2}\right)$$

$$\hat{w}_{ML} = \arg\min_w \sum_{i=1}^{N} (f_w(x_i) - y_i)^2$$

Laplace distribution Λ heavy tail

$$p(y|x,w) = \frac{1}{2b} \exp\left(-\frac{|y-\mu|}{b}\right)$$

$$\hat{w}_{ML} = \arg\min_w \sum_{i=1}^{N} |f_w(x_i - y_i)|$$

Classification Problems

Categorical distribution

$$\hat{w}_{ML} = \arg\max_w \sum_{i=1}^{N} \log\left(y_i | x_i, w\right)$$

$$= \arg\max_w \sum_{i=1}^{N} \log \prod_{c=1}^{C} f_w^{(c)}(x_i)^{y_{ic}}$$

$$= \arg\min_w \sum_{i=1}^{N} \boxed{\sum_{c=1}^{C} -y_{i,c} \log f_w^{(c)}(x_i)}$$

Cross-entropy loss

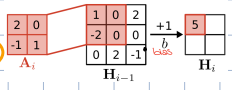
Softmax

$$\text{softmax}(x) = \left(\frac{\exp(x_1)}{\sum_{k=1}^{C}\exp(x_k)}, \dots, \frac{\exp(x_C)}{\sum_{k=1}^{C}\exp(x_k)}\right)$$

Convolutional neural network basics

Convolution layers

Correlation



bias

$$I'(i,j) = \sum_{m=-u}^{u}\sum_{n=-u}^{u} k(m,n) I(i+m, j+n)$$

Convolution (if $k(m,n) = k(-m,-n)$ then convolution = correlation)

$$I'(i,j) = \sum_{m=-u}^{u}\sum_{n=-u}^{u} k(-m,-n) I(i+m, j+n)$$

Fully connected

#Weights:

$H \times W \times C_{out} \times (H \times W \times C_{in} + 1)$

convolutional layer

#Weights:

$C_{out} \times (K \times K \times C_{in} + 1)$

Padding: add boundary of appropriate size with zeros around input tensor

Downsampling: reduces the spatial resolution; increases the receptive field (which pixels influence a neuron)

Stride: size of the jump

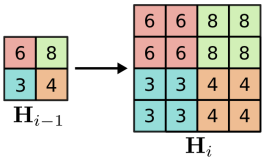
Pooling: has no parameters (typical pooling operations: max, min, mean)

Example: Filter: $K \times K$, Padding: P, Stride: S

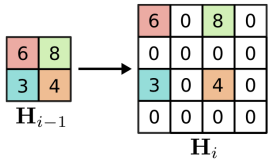
$$W_{out} = \left\lfloor \frac{W_{in} + 2P - K}{S} \right\rfloor + 1 \qquad H_{out} = \left\lfloor \frac{H_{in} + 2P - K}{S} \right\rfloor + 1$$

Upsampling: yields output at the same resolution as input

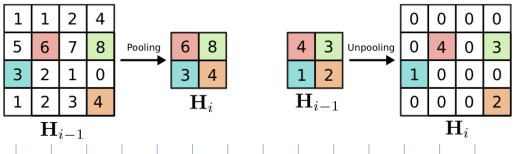
- nearest neighbor



- bed of nails



- max unpooling



LECTURE 5

Object recognition

Part-and-shape models

- Object as a set of parts
- relative locations between parts
- appearance of parts

History:

60-90: The geometric era
90: Appearance-based models
mid 90: Sliding window approaches
late 90: Local features
2000: Parts-and-shape models
mid 00: Bag-of features

Bag-of-features

Assumption: spatial information of local features can be ignored for object recognition.

An image (datapoint) is a vector of counts over each word (feature)

$$v_d = [n(w_{1,d})\alpha_1, n(w_{2,d})\alpha_2, \ldots]$$

$n(\cdot)$ = counts the number of occurrences
α_i = weights each word by a heuristic $= \log\left\{\frac{D}{\sum_{d'} 1[\text{word } i \text{ in } d']}\right\}$ → inverse document frequency

Pipeline:

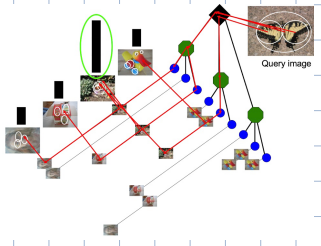
1. Feature extraction
   extracting features from images (regular grid v interest points) + SIFT descriptor
2. Dictionary learning
   learn visual words using clustering (K-means)
3. Encode
   build BOW vectors for each image
4. Classify
   Train and test data using BOWs

Pros
- flexible to geometry / deformations / viewpoint
- compact summary of image content
- provides fixed dimensional vector representing for sets
- very good results in practice

Cons
- background and foreground mixed
- optimal vocabulary formation remains unclear
- basic models ignore geometry

Precision: $\frac{tp}{tp+fp}$ Recall: $\frac{tp}{tp+fn}$

Requirement: sparsity



Specific object detection

Inverted index (word : images)
1. extract words in query
2. inverted file index to find relevant frames
3. compare/sort word-counts

Object Category Detection (object or not-object)

Object Recognition

1. Specify Object model
   what are the object parameters?
2. Generate hypotheses
   propose an alignment of the model to the image
3. Score hypotheses
   mainly gradient-based features, many classifiers
4. Resolve detection
   rescore each proposed object based on context information.

Sliding window Face detection with Viola-Jones

Slow training
Fast detection

1. Integral images for fast evaluation $I_\Sigma(x,y) = \sum_{\substack{x'\le x \\ y'\le y}} i(x',y')$
   - Rectangle filters repeated thousands times
     - how to chose right ones?
2. Boosting for feature selection (AdaBoost)

2-class classification problem

- Given: training set $X = \{x_1, \ldots, x_N\}$ with target values $T = \{t_1, \ldots, t_N\}$, $t_n \in \{-1, 1\}$.
- Associated weights $W = \{w_1, \ldots, w_N\}$ for each training point.

Basic steps

- In each iteration, AdaBoost trains a new weak classifier $h_m(x)$ based on the current weighting coefficients $W^{(m)}$.
- We then adapt the weighting coefficients for each point
  - Increase w_n if x_n was misclassified by $h_m(x)$.
  - Decrease w_n if x_n was classified correctly by $h_m(x)$.
- Make predictions using the final combined model

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right) \qquad h_m(\mathbf{x}) = \begin{cases} -1 & \text{if } f_m < \theta_m \\ +1 & \text{otherwise} \end{cases}$$

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,\ldots,N$.

2. For $m = 1,\ldots,M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) \qquad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$

   b) Estimate the weighted error of this classifier on X:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   c) Calculate a weighting coefficient for $h_m(x)$:

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$

   d) Update the weighting coefficients:

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\}$$

- Evaluate all selected weak classifiers on test data.

$$h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$$

- Final classifier is weighted combination of selected weak classifiers:

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

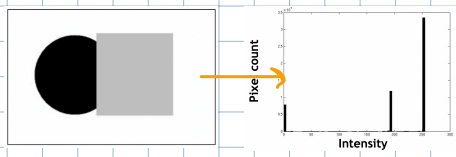
- Very simple procedure!

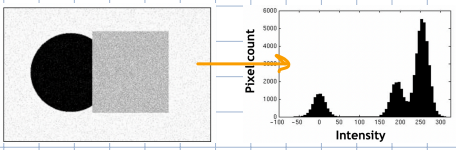
LECTURE 6

Segmentation as grouping

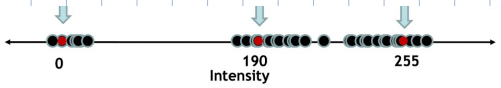
Goals:
- identify groups of pixels that belong together
- delineate object and background regions
- group similar-looking pixels for efficiency of further processing in "superpixels".

K-means:





To determine the three main intensities that define our groups we need to cluster. The three "centers" as the representative intensities

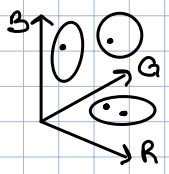


Algorithm:
1. randomly initialize the cluster centers $c_1, \dots, c_k$
2. Given cluster centers, determine points in each cluster
   - for each point p, find the closest c_i. Put p into cluster i
3. Given points in each cluster, solve for c_i
   - set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2.

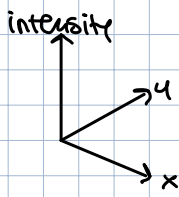
Properties:
- Always converge
- Might be stuck in local minimum of objective function

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$

Depending on what we choose as the feature space, we can group pixels in different ways.



← { feature space: color value (3D)
point $p = \begin{pmatrix} R = 15 \\ G = 189 \\ B = 2 \end{pmatrix}$

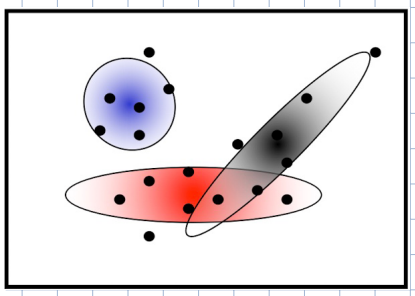


← { feature space: intensity + position similarity
point $p = \begin{pmatrix} X = \\ Y = \\ I = \end{pmatrix}$ (spatial coherence)

PROS	CONS
- Simple, fast to compute - Converge to local minimum of within-cluster squared error	- setting k? - sensitive to initial centers - sensitive to outliers - Detects spherical clusters only - Assuming mean can be computed

Mixture of gaussians (MoG)

Probabilistic clustering: instead of treating the data as a bunch of points, assume that they are all generated by sampling a continuous function. This function is called the generative model and is defined by vector of parameters θ.



In MoG the generative model is a mixture of gaussians. There will be k gaussian blobs $\sim N(\mu_b, V_b)$

$$p(x|\mu_b, V_b) = \frac{1}{\sqrt{(2\pi)^d |V_b|}} \cdot \exp\left(-\frac{1}{2}(x-\mu_b)^T V_b^{-1}(x-\mu_b)\right)$$

Blob b is selected with probability α_b.
The likelyhood of observing x is:

$$p(x|\theta) = \sum_{b=1}^{k} \alpha_b p(x|\theta_b) \qquad \theta = [\mu_1, \dots, \mu_k, V_1, \dots, V_k]$$

Goal: find θ that maximize the likelihood function overall all N datapoints

$$X = \{x_1, \dots, x_N\} \qquad p(X) = \prod_{x_i \in N} p(x_i|\theta)$$

EM - Approach:

- **E-step**
  - Compute probability that point x is in blob b, given current guess of θ

$$P(b|x, \mu_b, V_b) = \frac{\alpha_b P(x|\mu_b, V_b)}{\sum_{i=1}^{K} \alpha_i P(x|\mu_i, V_i)}$$

- **M-step**
  - Compute overall probability that blob b is selected

$$\alpha_b^{new} = \frac{1}{N}\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b) \qquad (N \text{ data points})$$

  - Mean of blob b

$$\mu_b^{new} = \frac{\sum_{i=1}^{N} x_i P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$

  - Covariance of blob b

$$V_b^{new} = \frac{\sum_{i=1}^{N} (x_i - \mu_b^{new})(x_i - \mu_b^{new})^T P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$

given current guess of blobs, compute probabilistic ownership of each point.

given ownership probabilities, update blobs to maximize likelihood function.

PROS

- Probabilistic interpretation
- Soft assignments between data points and clusters
- Generative model, can predict novel data points
- Relatively compact storage ($O(kd^2)$)

CONS

- Initialization (often a good idea to start from output of k-means.
- Local minima
- Need to know number of components k
- Need to choose blob generative model
- Numerical problems are often a nuisance

Mean-shift segmentation (model-free clustering)

Iterative mode search:

1. Initialize random seed center and window W
2. Calculate center of gravity ("the mean") of W: $\sum_{x \in W} x H(x)$
3. Shift the search window to the mean
4. Repeat steps 2+3 until convergence

Overall algorithm:

1. Choose features (color, gradients, texture, etc)
2. Initialize windows at individual pixel location (size h = scale of clustering)
3. Start mean-shift from each window until convergence
4. Merge windows that end up near the same "peak" or mode

PROS

- General, application-independent tool
- Model-free, does not assume any prior shape on data clusters
- Just a single parameter (window size h)
- Finds variable number of modes given the same h
- Robust to outliers

CONS

- Output depends on window size h
- Window size selection is not trivial
- Computationally expensive
- Does not scale well with dimension of feature space

Hough transforms (edge-based)

Uses the structure of shapes to extract them in a parameter space of limited dimension.

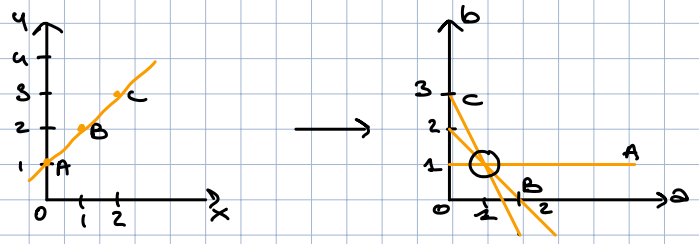
Detecting straight lines

straight line: $y = ax + b \longrightarrow b = (-x)a + y$

1. the parameter space is discretized
2. a counter is increased at each cell where a line pass
3. peaks are detected

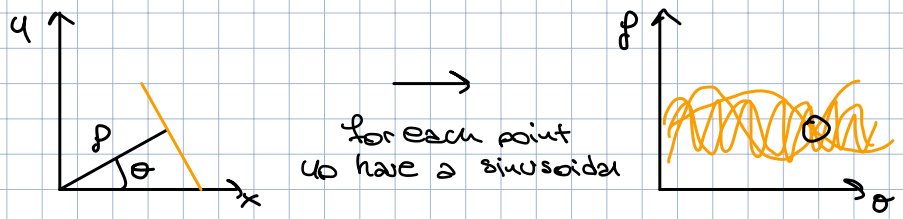
The Hough transform:

1. Discretize an edge pts
2. For each point draw the above line in (a,b) parameter space



⚠ cannot represent vertical lines ($a = \infty$)

straight line: $x\cos\theta + y\sin\theta = \rho$



Remark

1. Time consuming
2. Robust to outliers and lots of points not on a line
3. Peak detection is difficult with noise in edge coordinates.
4. Robustness of peak detection increased by weighting contributions (votes that are proportional with intensity gradients magnitude).
5. Can do it also for circles and even higher-order parametric shapes (but it quickly gets harder as number of shape parameters grows).

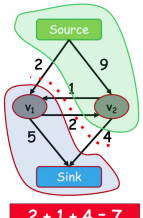
GraphCut

In graph-cut, computing the disparity is formulated as a graph labeling problem. Each pixel corresponds to a graph node and each disparity to a label. The goal is to find a labeling $f : P \to L$ which minimizes the energy function

$$E(f) = E_{data}(f) + \lambda E_{smooth}(f) = \sum_{p \in P} \underbrace{D_p(f_p)}_{\text{unary potential}} + \lambda \sum_{p,q \in \mathcal{N}} \underbrace{S(f_p, f_q)}_{\text{pairwise potential}}.$$

P represents the set of pixels and L represents a discrete set of labels corresponding to different disparities. $D_p(f_p)$ is the cost of assigning label f_p to pixel p and it is given by the SSD value calculated for the disparity corresponding to label f_p. $\mathcal{N}$ is the set of neighboring pixels and $S(f_p, f_q)$ is the cost of assigning labels f_p and f_q to neighboring pixels p and q.

The s-t-Mincut Problem



2 + 1 + 4 = 7

What is an st-cut?

An st-cut (S,T) divides the nodes between source and sink.

What is the cost of a st-cut?

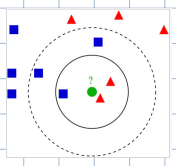
Sum of cost of all edges going from S to T

What is the st-mincut?

st-cut with the minimum cost

Learning-based approaches

K-nearest neighbors (KNN)



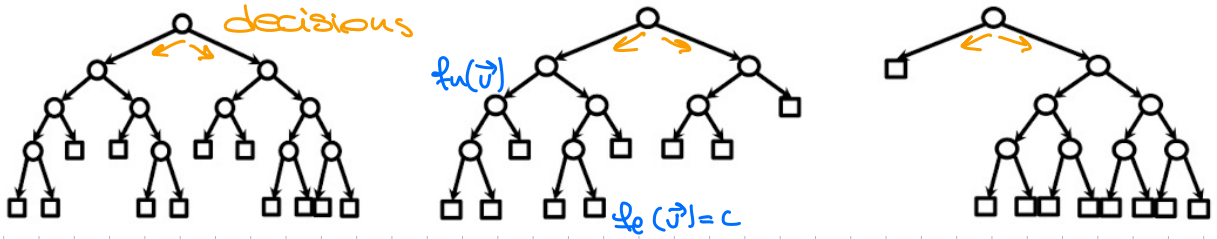
PROS

- Very simple to implement / understand
- Efficient implementations possible for approx. NNs
- Distance definition is flexible

CONS

- Highly depends on the definitions and K
- Need to keep the entire data in memory for distance computation
- For high dimensional problems (with high d) might need many training samples for accuracy.
- Other methods have better generalization ability

Random Forest



Training:
1. deciding on the (binary) node tests
2. deciding on when to stop splitting.

Randomness:
1. node tests result from randomly generated sets of candidates tests.
2. another aspect of the randomness is that different trees can be trained with different random subsets.

Stopping:
1. all samples belong to a class
2. too few samples

PROS

- Easy to implement
- Very efficient during testing
- Can easily use diverse features
- Can handle high dimensional spaces

CONS

- Lots of parametric choices
- Need large number of data
- Training can take time

LECTURE 7

Image segmentation evaluation → Mean intersection over union

$$\frac{1}{C}\sum_{i}^{C}\frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}$$

C = number of categories
n_{ij} = the number of pixels of class i predicted to belong to class j
t_i = total number of pixels of class i.

Transfer training:

STEP 1: Learn the filters on a related task and dataset (pre-training)

STEP 2: Learn on the target problem (fine-tuning)
- initialize the model with the pre-trained filters in Step 1 as much as possible
- train the model with BP and smaller learning rate
- adjust the filters instead of learning them from scratch

Semantic segmentation:

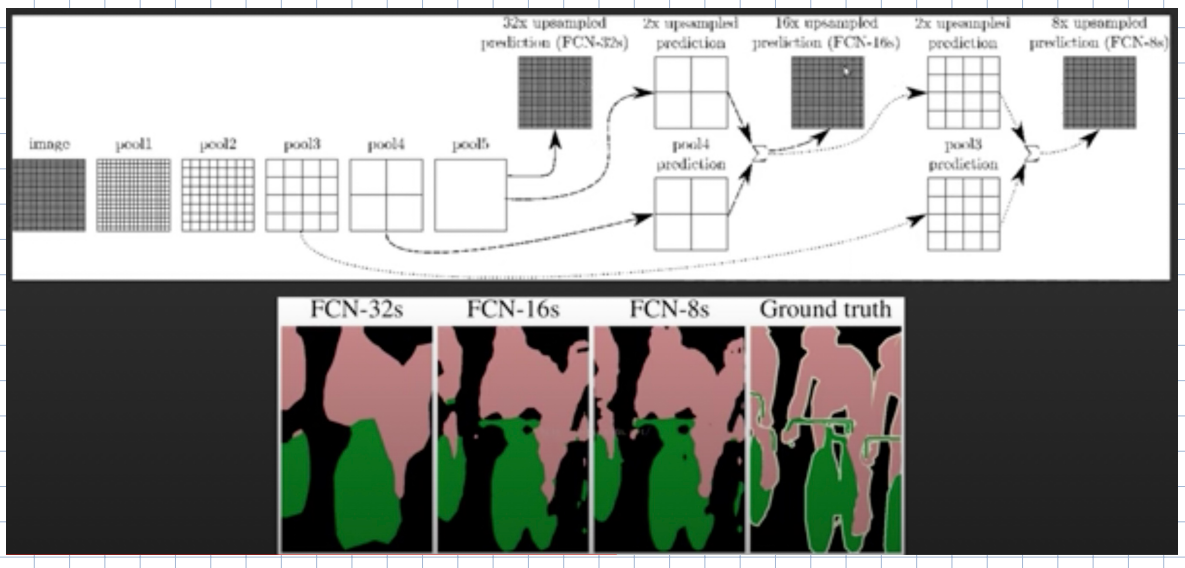
Hyper-column

- Viable way to transfer the image classification models to segmentation
- Reasonable results, but coarse boundries
- Depends on classical methods, like region proposals based on pixel grouping

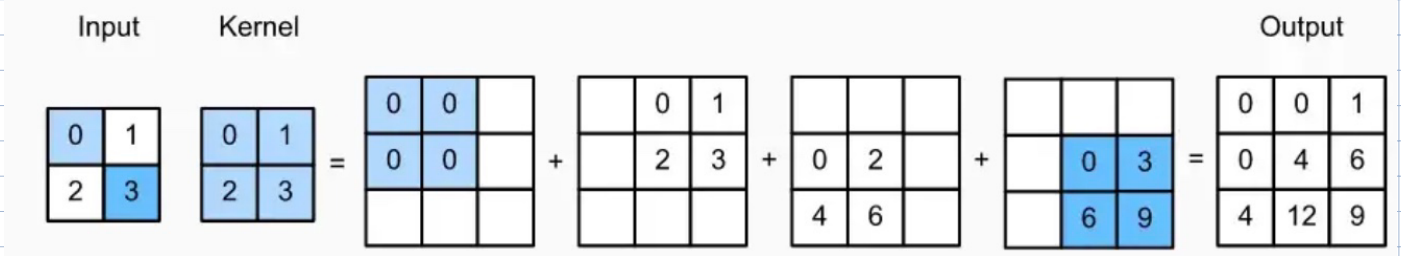
1. For each pixel location, take the activations from each level
2. Merge the feature together as the representation for the pixel
3. Multiscale representation (hypercolumn)

- Pick representative levels
- Additional convolutional layers to add capacity and change channel numbers
- Use bilinear interpolation to upsample the feature maps
- Sum the upsampled feature maps.

Fully convolutional networks



Transposed convolutions (deconvolutions) → use to upsample FCN



Fully connected Conditional Random Field

$$E(L) = \sum_p E_p(L_p) + \sum_{p,q \in N} E(L_p, L_q)$$

PROS

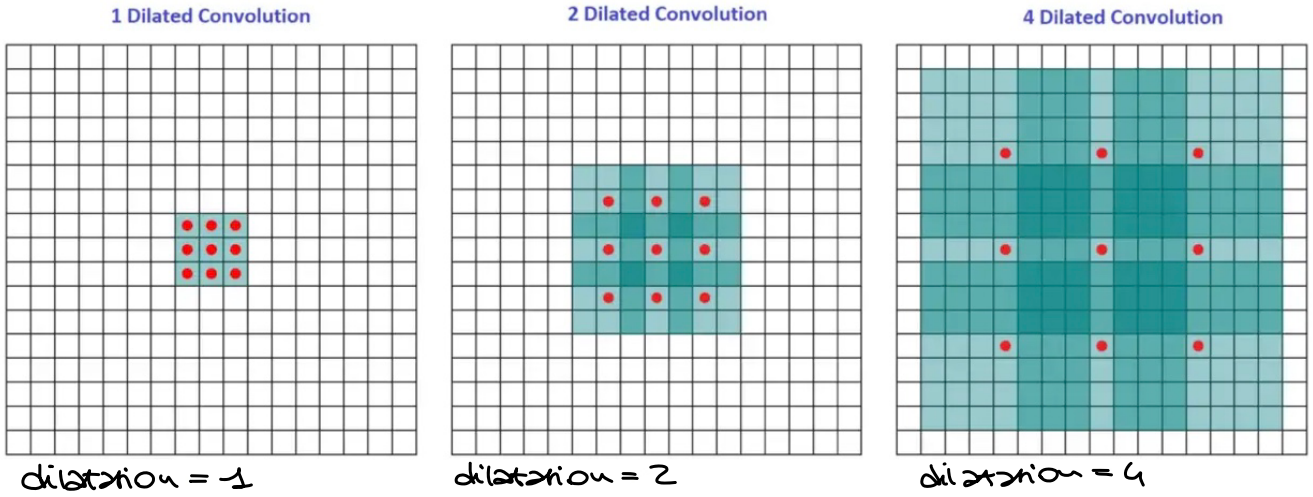
- Very effective method
- The results look super sharp

CONS

- A post-proc processing method
- Many tuned parameters

Dense prediction models

Dilated convolutions



Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

LECTURE 8

Detection via classification

Pipeline:
1. Obtain training data
2. Define features (HoG)
3. Train a classifier
4. Use sliding window (AdaBoost)

Generalized Hough Transform (ISM – implicit shape model)

Basic ideas:
1. Learn an appearance codebook
   - Extract local features at interests points
   - Agglomerative clustering => codebook
2. Learn a star-topology structural model (spatial distribution)
   - Feature are considered independent given obj. center
   - Match codebook to training images
   - Record matching positions on object

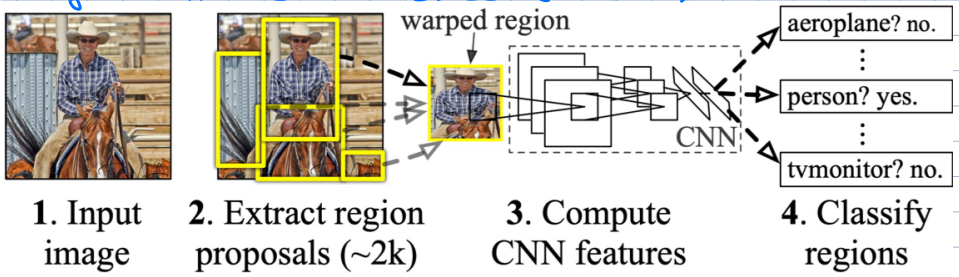
A visual vocabulary is used to index votes for object position (a visual word = "a part"). Objects are detected as consistent configurations of the observed parts (visual words).

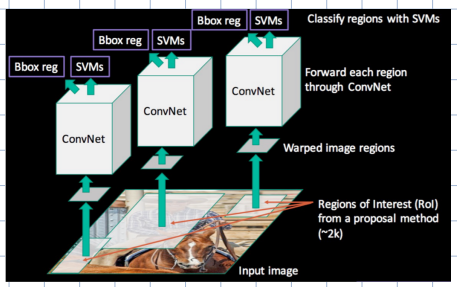
PROS
- Works well for many different object categories
- Flexible geometric model (can recombine parts seen on different training examples.
- Learning from few (50–100) training examples
- Optimized for detection, good localization properties

CONS
- Needs supervised training data
  - Object bounding boxes for detection
  - Segmentations for top-down segmentation
- Only weak geometric constraints
  - result segmentations may contain superfluous body parts
- Purely representative model
  - No discriminative learning

Object detection with deep learning

Regions with CNN features (R-CNN)





Classification is made with "Linear model with class-dependent weights"

Linear SVM → support vector machine

$f_c(x_{fc}) = w_c^T x_{fc}$

$\begin{cases} x_{fc} = \text{features from the network} \\ c = \text{object class} \end{cases}$

Prediction of the 2D box

- Necessary, since the proposal region might not fully coincide with the (annotated) object bounding box
- Perform regression for location (x^*, y^*), width w^* and height h^*

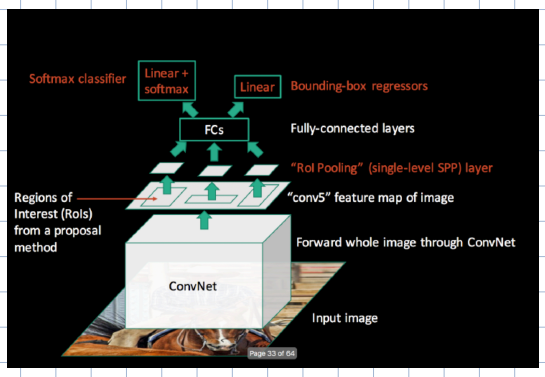
$$\frac{x^* - x}{w} = W_{c,w}^T x_{pool5} \qquad \frac{y^* - y}{h} = W_{c,w}^T x_{pool5} \qquad \ln \frac{w^*}{w} = \ln \frac{h^*}{h} = W_{c,w}^T x_{pool5}$$

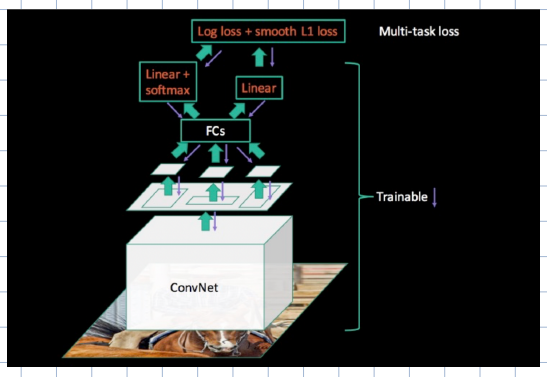
where x_{pool5} are the features from pool5 layer of the network

CONS

- Ad hoc training objectives
  - fine tuning network with softmax classifier (log loss)
  - train post-hoc linear SVMs (hinge loss)
  - train post-hoc bounding-box regressor (squared loss)
- Training (3 days) and testing (47s x image) is slow.
- Takes a lot of disk space: need to store all precomputed CNN features for training the classifiers

Fast R-CNN



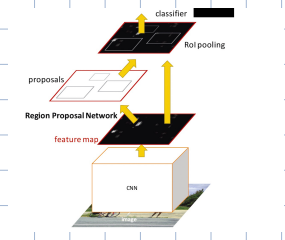


Region Proposal Networks (RPN)

- Remove dependence on external region proposal algorithm.
- Instead, infer region proposals from same CNN

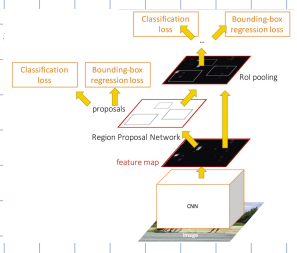
=> feature sharing

=> object detection in a single pass becomes possible

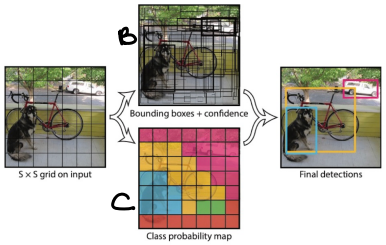


Faster R-CNN = Fast R-CNN + RPN

- Region proposal goes deeper
- Uses RoI (Region of Interest) pooling and anchors to extract region features.
- One network → 4 losses (joint training)



YOLO: You only look once

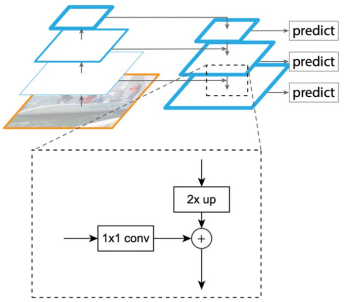


- YOLO divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
- Each grid cell predicts B bounding boxes and confidence scores for those boxes.
- Each bounding box consists of 5 predictions x, y, w, h and confidence.
- Each grid cell also predicts C conditional class probabilities.
- What's the output tensor size?

1. Find the box with greatest confidence
2. Remove the other boxes with IoU ≥ thresh
3. If there are other boxes, go to 1.

Feature pyramid Networks (FPN)

- A top-down architecture with skip connections. FPN model leverages it as a feature pyramid, with predictions made independently at all levels.
- With a coarser-resolution feature map, FPN upsamples the spatial resolution by a factor of 2 using nearest neighbor upsampling. The two feature maps are merged by addition.



FPN on Faster RCNN

- Assign anchors of a single scale to each level.
- There are 5 levels in total, where the last level feature is simply a stride two subsampling of the last layer of ResNet.
- Use anchors of multiple aspect ratios {1:2, 1:1, 2:1} at each level.
- Assign training labels to the anchors based on their Intersection-over-Union (IoU) ratios with ground-truth bounding boxes.
- An anchor is assigned a positive label if it has the highest IoU for a given ground truth box or an IoU over 0.7 with any ground-truth box, and a negative label if it has IoU lower than 0.3 for all ground-truth boxes.

LECTURE 9

Track a point

Optimize displacement for this energy, so the two pixels have the same color.

$$E(h) = [I_0(x+h) - I_1(x)]^2$$

$$E(h) \approx [I_0(x) + hI_0'(x) - I_1(x)]^2$$

$$\frac{\partial E}{\partial h} = 2I_0'(x)[I_0(x) + hI_0'(x) - I_1(x)]$$

$$\frac{\partial E}{\partial h} = 0 \rightarrow h \approx \frac{I_1(x) - I_0(x)}{I_0'(x)}$$

Problem 1: $I_0'(x) = 0$ if the image gradient is zero (the image region is flat)

Problem 2:
- Motion to closest minimum has to be assumed
- Indirect result: frame-rate should be faster than motion ($\frac{1}{2}$ wavelength)
- Non-convex minimum may indicate multiple solutions

Solution:
- optical flow in motion estimation
- Harris detector

Track a bigger box

LUCAS-KANADE - Template tracking

Keep a template image to compare with each frame (only for small patches).

$$E(u,v) = \sum_{x,y} [I(x+u, y+v) - T(x,y)]^2 \quad T(x,y) = \text{template}$$

$$= \sum_{x} [I(x+\Delta x) - T(x,y)]^2 \quad W(x,p) = \text{optimal warp}$$

$$\Rightarrow \sum_{x} [I(W(x;p+\Delta p)) - T(x)]^2 \quad \text{taylor expansion}$$

$$= \sum_{x} [I(W(x;p)) + \nabla I \frac{\partial W}{\partial p} \Delta p - T(x)]^2$$

$$\Delta p = H^{-1} \sum_{x} [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))] \quad \text{set derivative to 0}$$

$$\text{where} \quad H = \sum_{x} [\nabla I \frac{\partial W}{\partial p}]^T [\nabla I \frac{\partial W}{\partial p}]$$

Warp I to obtain $I(W([x\ y];P))$

Compute the error image $T(x) - I(W([x\ y];P))$

Warp the gradient ∇I with $W([x\ y];P)$

Evaluate $\frac{\partial W}{\partial P}$ at $([x\ y];P)$ (Jacobian)

Compute steepest descent images $\nabla I \frac{\partial W}{\partial P}$

Compute Hessian matrix $\sum (\nabla I \frac{\partial W}{\partial P})^T (\nabla I \frac{\partial W}{\partial P})$

Compute $\sum (\nabla I \frac{\partial W}{\partial P})^T (T(x,y) - I(W([x,y];P)))$

Compute ΔP

Update $P \longleftarrow P + \Delta P$

PROS
- Beautiful framework
- It can handle different parameter space
- It can converge fast in a high-frame rate video

CONS
- Not robust to image noise or large displacement
- Some trasformations are impossible to parameterize

Track by detection

Tracking by features

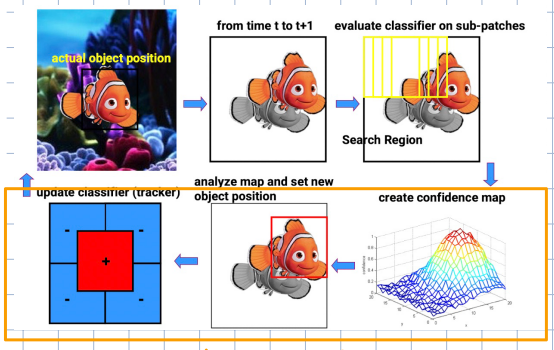
1. Detect key points (in reference and image)
2. Build feature descriptors
3. Match key points descriptors
4. Geometric verification (robust 3D pose calculation - RANSAC)

Acceleration:
- Tree structure (KD-tree)
- Approximate nearest neighbor
- Hashing
- Parallel implementation

Known object category

1. Detect objects independently in each frame
   - usually supervised learning (ConvNets)
2. Associate detections over time into tracks

Online learning

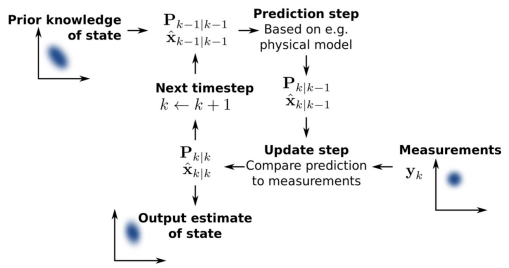


gradual drift

To avoid drift:
- Only thing we're sure about the object is its initial model
- We can "anchor" (correct our model with this information in order to help avoid drift

Motion

Kalman filter



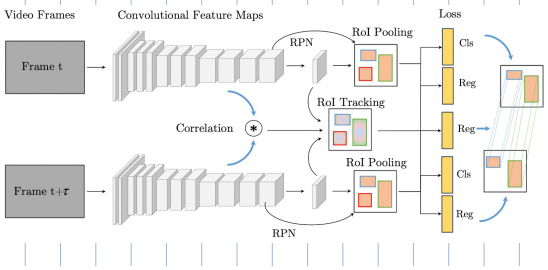
example: $x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$

- u, v = bounding box coordinate
- s = area of the bounding box
- r = aspect ratio

Training: neural networks
→ network learns generic object tracking

Test: neural network (frozen weights)
→ network tracks novel objects (no fine-tuning)

Combining tracking and detection



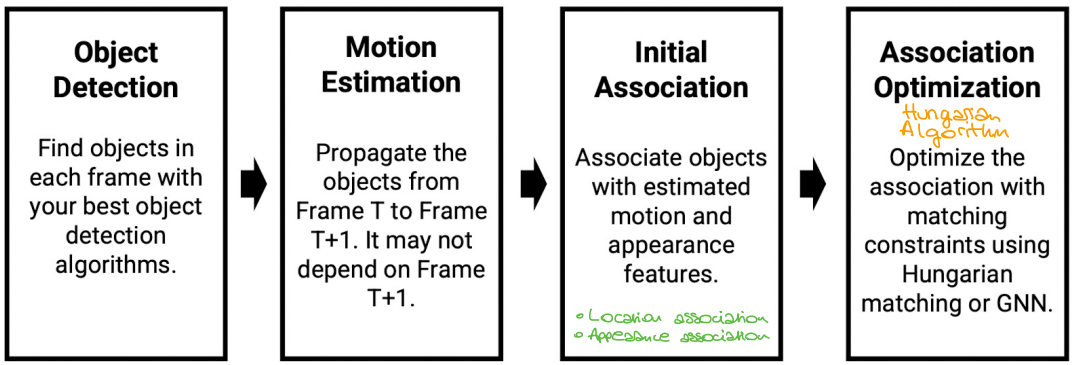
RoI pooling

1. Extract features with downsampling on input image (create a grid)
2. RoI is rounded to image grid
3. RoI is broken in grid.
4. MaxPool in each RoI cell.

RoI Align

1. Extract features with downsampling on input image (create a grid)
2. RoI is broken in grid.
3. RoI is broken in grid with same resolution of the grid
4. For each RoI cell, bilinear interpolate the features
5. MaxPool in each RoI cell.

Multiple object tracking



Why doesn't appearance provide enough information in current models?

- similar bounding boxes.
- misleading regions in the background.

LECTURE 10

Projective geometry

Point on a line:	$\ell^T x = 0$
Line through two points:	$\ell = x_1 \times x_2$
Point on a plane:	$\vec{\pi}^T x = 0$
Transformation of points:	$x' = Hx$
Transformation of lines:	$\ell' = H^{-T}\ell$

Conic | locus of points m satisfying a homogeneous quadratic equation:
$$S(m) = m^T C m = 0$$
where C is a 3×3 symmetric matrix defined up to scale (#dof = 5)

Inhomogeneous: $ax^2 + bxy + cy^2 + dx + ey + f = 0$ (#dof = 5)

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Dual conic: | locus of lines satisfying a homogeneous quadratic equation:
$$\ell^T C^* \ell = 0$$
where C is a 3×3 symmetric matrix defined up to scale

Line tangent to a conic in point m: $\ell = C^T m = Cm$
Lines tanget to a conic: $\ell^T C^* \ell = 0$ if full rank $C^* = C^{-1}$
Transformations of a conic: $C' = H^{-T} C H^{-1}$ $C^* = H C^* H^T$ $(C')^* = (C^*)'$

Degenerate conic: | Matrix C is not full rank
- $C = \ell m^T + m \ell^T$ (ℓ, m are two lines $x \in \ell, x \in m$)
- $C = \ell \ell^T$ (ℓ is a line $x \in \ell$)

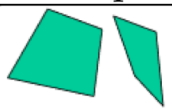
For degenerate conics $(C^*)^* \neq C$

Quadric: | locus of all points M satisfying a homogeneous quadratic equation:
$$M^T Q M = 0$$
where Q is a 4×4 symmetric matrix defined up to scale

Dual quadric: | locus of all planes $\vec{\pi}$ satisfying a homogeneous quadratic equation
$$\vec{\pi}^T Q^* \vec{\pi} = 0$$
where Q is a 4×4 symmetric matrix defined up to scale

Plane tangent to a quadric in point M: $\vec{\pi} = Q^T M$
Planes tangent to a quadric $\vec{\pi}^T Q^* \vec{\pi} = 0$ if full rank $Q^* = Q^{-1}$
Transformation of a quadric: $Q' = T^{-T} Q T^{-1}$ $Q^{*\prime} = T^T Q T$ $(Q')^* = (Q^*)'$

Cross ratio: | invariant property under projective transformations.
Assume 4 points M_1, M_2, M_3, M_4 are collinear. They can be expressed as $M_i = M + \lambda_i M'$
$$CR\{M_1, M_2, M_3, M_4\} = \frac{\lambda_1 - \lambda_3}{\lambda_1 - \lambda_4} : \frac{\lambda_2 - \lambda_3}{\lambda_2 - \lambda_4}$$

Transformation	2D DoF	3D DoF	Example	INVARIANTS
Projective	8	15		- Concurrency - collinearity - order of contact - cross ratio
Affine	6	12		- line at infinity / plane at infinity - parallelism - ratio of lengths on ll lines - linear combination of vectors - ratio of areas
Similarity	4	7		- ratio of lengths - angles - circular points $\begin{cases}(1,i,0):I\\(1,-i,0):J\end{cases}$ - dual conic $C^*_\infty: (IJ^T + JI^T)$ - absolute conic - absolute dual quadric
Euclidian	3	6		- lengths - areas

rotation = 1 dof rotation = 3 dof

Projectivity: invertible mapping from P^2 to itself such that three points x_1, x_2, x_3 lie on the same line if and only if $h(x_1), h(x_2), h(x_3)$ do.

Theorem: a mapping $h: P^2 \to P^2$ is a projectivity if and only if there exist a non-singular 3×3 matrix such that for any point in P^2 represented by a vector x it is true that $h(x) = Hx$.

projectivity = collineation = homography

Singular value decomposition:

method used to solve $\min \|Ax\|$ subject to $\|x\| = 1$

$A = U \Sigma V^T$ $\begin{cases} U = \text{rotation matrix} \\ \Sigma = \text{scaling matrix (diagonal positive decreasing values)} \\ V^T = \text{rotation matrix} \end{cases}$

Solution $\rightarrow$ $A = V_n$: last column of matrix V

Gold Standard algorithm

In P^2, projectivity matrix is 3×3 up to scale $\Rightarrow$ 8 dof.
Each measurement gives 2 equation $\Rightarrow$ only 4 points needed.
If you have more, you cannot have exact solution because of the noise $\Rightarrow$ need to find best according to cost function.

- cost function is optimal for some assumptions
- algorithm that minimizes the cost function.

Direct linear transformation (DLT)

$x_i' = Hx_i$ → only if up to scale. But always ∥

$x_i' \times Hx_i = 0$ → cross product

$$x_i' = (x_i', y_i', w_i')^T \qquad Hx_i = \begin{pmatrix} h_1 x_i \\ h_2 x_i \\ h_3 x_i \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

$$x_i' \times Hx_i = \begin{pmatrix} y_i' h_3^T x_i - w_i' h_2^T x_i \\ w_i' h_1^T x_i - x_i' h_3^T x_i \\ x_i' h_2^T x_i - y_i' h_1^T x_i \end{pmatrix} = \begin{pmatrix} 0 & -w_i' x_i & y_i' x_i \\ w_i' x_i & 0 & -x_i' x_i \\ -y_i' x_i & x_i' x_i & 0 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = A_i h = 0$$

only 2 independant equations → you can drop 3rd line.
If you have more than 4 points, minimize $\|Ah\| \to 0$.

Objective

Given n≥4 2D to 2D point correspondences $\{x_i \leftrightarrow x_i'\}$, determine the 2D homography matrix H such that $x_i' = Hx_i$

Algorithm

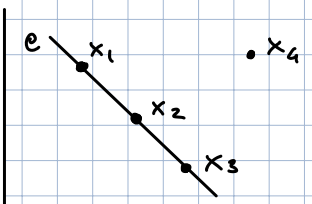
(i) For each correspondence $x_i \leftrightarrow x_i'$ compute A_i. Usually only two first rows needed.

(ii) Assemble n 2x9 matrices A_i into a single $2n$x9 matrix A

(iii) Obtain SVD of A. Solution for h is last column of V

(iv) Determine H from h

Degenerate configuration



Costraints: $x_i' \times Hx_i = 0$

Define: $H^* = x_4' e^T$ (H is rank-1, not homography)

Then: $\begin{cases} H^* x_i = x_4' e^T x_i = 0 & i = 1,2,3 \\ H^* x_i = x_4' e^T x_4 = k x_4' & i = 4 \text{ (maps to itself)} \end{cases}$

If H^* is unique solution, no homography.
If there is further solution H, then $\alpha H^* + \beta H$ is solution

Cost functions

- Algebraic distance
- Geometric distance
- Reprojection error

Algebraic distance

$e = Ah$

$d_{alg}(x_i', Hx_i)^2 = \|e_i\|^2$

$\sum_i d_{alg}(x_i', Hx_i)^2$

$d_{alg}(x_1, x_2)^2 = a_1^2 + a_2^2$ where $a = (a_1, a_2, a_3)^T = x_1 \times x_2$

Geometric distance

$\hat{H} = \underset{H}{\operatorname{argmin}} \sum_i d(x_i', H\tilde{x}_i)^2 = d_{alg}(x_i', Hx_i)/(w_i' H w_i)$ euclidean distance

Symmetric error

$\hat{H} = \underset{H}{\operatorname{argmin}} \sum_i d(x_i, H^{-1}x_i')^2 + d(x_i', Hx_i)^2$

Reprojection error

$(\hat{H}, \hat{x}_i, \hat{x}_i') = \underset{(H, \hat{x}_i, \hat{x}_i')}{\operatorname{argmin}} \sum_i d(x_i, \hat{x}_i)^2 + d(x_i', \hat{x}_i')^2$

Mahalanobis error: $\begin{cases} \sum_i \|x_i - \bar{x}_i^2\|_\Sigma + \|x_i' - \bar{x}_i'^2\|_{\Sigma'} \\ \text{where } \|X - \bar{X}\|_\Sigma^2 = (x - \bar{x})\Sigma^{-1}(x - \bar{x}) \end{cases}$

measurement x with covariance Σ

Objective

Given n≥6 2D to 3D point correspondences $\{X_i \leftrightarrow x_i'\}$, determine the Maximum Likelyhood Estimation of P

Algorithm

(i) Linear solution:

(a) Normalization: $\tilde{X}_i = UX_i \quad \tilde{x}_i = Tx_i$

(b) DLT

(ii) Minimization of geometric error: using the linear estimate as a starting point minimize the geometric error:

$$\min_P \sum_i d(\tilde{x}_i, \tilde{P}\tilde{X}_i)^2$$

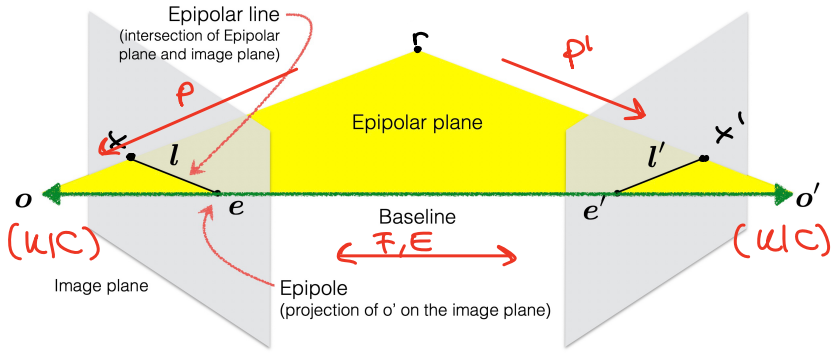
(iii) Denormalization: $P = T^{-1}\tilde{P}U$

Calibration

1. Canny edge detection
2. Straight line fitting to the detected edges
3. Intersecting the lines to obtain the images corners

LECTURE 11

Epipolar geometry



$$\begin{cases} e' = Fx \\ e = F^T x' \end{cases}$$

$$\begin{cases} e'^T F = 0 \\ Fe = 0 \end{cases}$$

every x is normalized!

$$\begin{cases} x = Pr \\ x' = P'r \end{cases}$$

Camera geometry

Given a set of corresponding image points $\{x_i \leftrightarrow x_i'\}$, what are the camera P and P' for the two viewers?

$(P, P') \rightarrow F$ unique : $\begin{cases} x = Pr \\ x' = P'r \\ x'^T = Fx \end{cases}$ $\begin{cases} P = [I|0] \\ P' = [M|m] \\ F = [m]_x M \end{cases}$

$F \rightarrow (P, P')$ NOT unique: $\begin{cases} \text{CANONICAL WAY} \\ P = [I|0] \\ P' = [M|m] = [e']_x F + e' v^T | \lambda e'] \end{cases}$ $F = [e']_x P' P^+ = \lambda F$

FUNDAMENTAL MATRIX:

The fundamental matrix satisfies the condition that for any pair of corresponding points $x \leftrightarrow x'$ in two images

$$\boxed{x'^T F x = 0}$$

- F is a unique 3x3 rank 2 matrix
- if $(P, P') \rightarrow F$, then $(P', P) \rightarrow F^T$
- $e'^T F x = 0 \ \forall x \Rightarrow e'^T F = 0$ $\quad F^T x' = 0 \ \forall x' \Rightarrow F^T = 0$
- F has 7 dof (3x3 - 1 (homogeneous) - 1 (rank 2))
- F is a correlation from a point to a line (not proper correlation, not invertible).
- General motion $F = [e']_x P' P^+ = [e']_x H_\pi$
- Pure traslation $F = [e']_x$ (only 2 dof)

The fundamental matrix is a generalization of the essential matrix, where the assumption of Identity matrices is removed.

ESSENTIAL MATRIX:

The essential matrix operates on image points expressed in 2D coordinates expressed in the camera coordinate system:

- $E e' = 0$, $E^T e = 0$
- E is singular
- E has two equal non-zero singular values.
- $E = [t]_x R$

Both E and F matrices completely describe the geometric relationship between corresponding points of a stereo pair of cameras. The only difference is that the former deals with calibrated cameras, while the latter with uncalibrated cameras.

$$F = K^{-T} E K^{-1} \leftrightarrow E = K^T F K$$

PROJECTIVE RECONSTRUCTION THEOREM:

If F is same for (P, P') and $(\tilde{P}, \tilde{P}')$, then there exists a projective transformation H so that $\tilde{P} = HP$ and $\tilde{P}' = HP'$.

$\Rightarrow$ If a set of point correspondences in two views determine the F uniquely, then the scene and the cameras may be reconstructed from these correspondences alone, and any two such reconstruction from these correspondences are projectively equivalent.

Canonical cameras given F: $P = [I|0]$, $P' = [[e']_\times F + ev^T \mid \lambda e']$

COMPUTATION of F:

- Linear 8-point
- Minimal 7-point
- Non-linear refinement

- Calibrated 5-point
- Calibrated + know vertical 3-point

8-POINT ALGORITHM

1. Normalize points ($x_{normalized} = K^{-1} x_{un\text{-}normalized}$)
2. Construct $M \times 9$ matrix A (M = # correspondences, $9 = f_1, f_2, \dots f_9$) $x'Fx = 0$
3. Find the SVD of A^TA
4. Entries of F are the elements of column of V corresponding to least singular value
5. Enforce rank 2 constraint on F $\rightarrow$ rank k: enforce all but the larger k values of SVD Σ to be 0.
6. Un-normalize $F = K'^{-T} F K^{-1}$

TRIANGULATION:

given (x, x'), (P, P'), which 3D point match to x?

$$\begin{cases} x = Pr \\ x' = P'r \end{cases} \xrightarrow{\text{scale factor}} x \times Pr = 0$$

cross product of two vectors of same direction is zero

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} p_1^T r \\ p_2^T r \\ p_3^T r \end{bmatrix} = \begin{bmatrix} y\, p_3^T r - p_2^T r \\ p_1^T r - x\, p_3^T r \\ x\, p_2^T x - y\, p_1^T r \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

only 2 independent equations

$$\begin{bmatrix} y\, p_3^T r - p_2^T r \\ p_1^T r - x\, p_3^T r \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} y\, p_3^T - p_2^T \\ p_1^T - x\, p_3^T \\ y'\, p_3'^T - p_2'^T \\ p_1'^T - x'\, p_3'^T \end{bmatrix} r = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$Ax = 0$ SVD!

RECONSTRUCTION:

1. Compute the Fundamental Matrix F from points correspondences (8-point algorithm)
2. Compute the camera matrices P from the Fundamental matrix

$P = [I|0]$ and $P' = [[e'_x]F | e']$

3. For each point correspondence, compute the point r in 3D space (triangulation)

DLT with $\begin{cases} x = Pr \\ x' = P'r \end{cases}$

MULTI-VIEW GEOMETRY

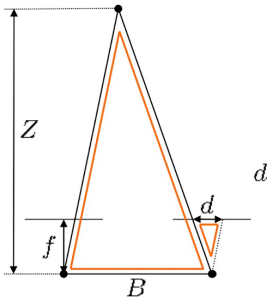
STRUCTURE FROM MOTION

Recovering 3D structure of the scene from camera (or object) motion

LECTURE 13

STEREO VISION

disparity = $x - x' = d = -\frac{bf}{z}$



$\frac{B}{Z} = \frac{d}{f}$

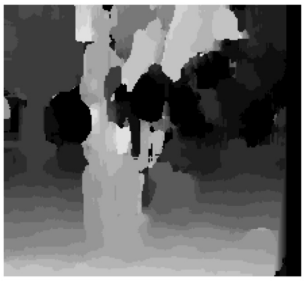
$d = -\frac{Bf}{Z}$

BASIC STEREO ALGORITHM

- For each epipolar line
  - For each pixel in the left image
    - compare with every pixel on the same epipolar line in the right image
    - pick pixel with minimum match cost

→ IMPROVEMENT: match windows eg. Lukas-Kanade

W = 3



W = 20

SMALLER WINDOW:
more detail (+)
more noise (−)

LARGER WINDOW:
smoother disparity maps (+)
less detail (−)

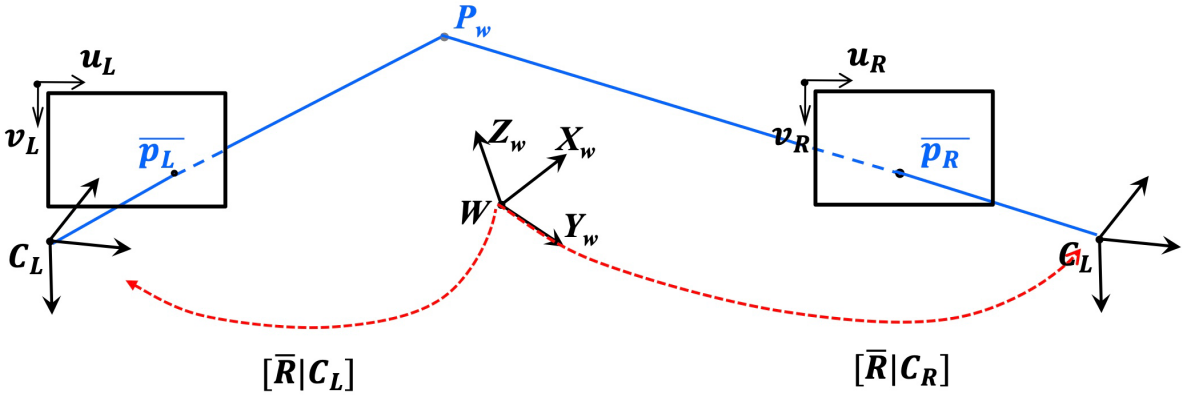
STEREO IMAGE RECTIFICATION

The goal of stereo rectification is to warp the left and the right camera images such that their focal planes are coplanar and their intrinsic parameters are identical.

- reproject image planes onto common plane parallel to line between optical centers
- a homography (3x3 transform) applied to both input images
- pixel motion is horizontal after this transformation

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R | T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \longrightarrow \underbrace{\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}}_{\text{camera point}} = K R^{-1} \left(\underbrace{\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}}_{\text{3D point}} - T \right)$$

Old Left camera	New Left camera	Old Right camera	New Right camera
$\lambda_L \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} = K_L R_L^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - C_L \right)$	$\rightarrow \quad \bar{\lambda}_L \begin{bmatrix} \bar{u}_L \\ \bar{v}_L \\ 1 \end{bmatrix} = \bar{K} \bar{R}^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - C_L \right)$	$\lambda_R \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} = K_R R_R^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - C_R \right)$	$\rightarrow \bar{\lambda}_R \begin{bmatrix} \bar{u}_R \\ \bar{v}_R \\ 1 \end{bmatrix} = \bar{K} \bar{R}^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - C_R \right)$



By solving for $\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$ for each camera, we can compute the homography (or warping) that needs to be applied to rectify each camera image.

$$\bar{\lambda}_L \begin{bmatrix} \bar{u}_L \\ \bar{v}_L \\ 1 \end{bmatrix} = \lambda_L \underbrace{\bar{K} \bar{R}^{-1} R_L K_L^{-1}}_{\text{Homography Left Camera}} \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix}$$

$$\bar{\lambda}_R \begin{bmatrix} \bar{u}_R \\ \bar{v}_R \\ 1 \end{bmatrix} = \lambda_R \underbrace{\bar{K} \bar{R}^{-1} R_R K_R^{-1}}_{\text{Homography Right Camera}} \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix}$$

How to choose $\bar{K}$, $\bar{R}$?

- $\bar{K} = (K_L + K_R)/2$
- $\bar{R} = [\bar{r}_1, \bar{r}_2, \bar{r}_3]$ where

$$\begin{cases} \bar{r}_1 = \dfrac{C_2 - C_1}{\| C_2 - C_1 \|} \\ \bar{r}_2 = r_3 \times \bar{r}_1 \\ r_3 = \bar{r}_1 \times \bar{r}_2 \end{cases}$$