UZ summary

Summary of the Machine Perception course lectures at FRI

ib8548

January 18, 2023

Contents

1 Image processing

Processing steps:

1. Convert gray image to binary image **(thresholding)**
2. Clean binary image **(morphologic filtering)**
3. Extract individual regions **(connected components)**

1.1 Thresholding

Otsu's algorithm:

1. Separate the pixels into two groups by intensity threshold T
2. For each group get an average intensity and calculate $\sigma^2_{between}$
3. Select the T that maximizes the variance:
   $T^* = argmax_T[\sigma^2_{between}(T)]$

1.2 Cleaning the image

1.2.1 Fitting and hitting

Fitting: all "1" pixels in the SE (structuring element) cover all "1" pixels in the image
Hitting: at least one "1" pixel in the SE covers a "1" pixel in the image

1.2.2 Erosion

- Reduces the size of structures
- Removes bridges, branches, noise
- $g(x,y) = \begin{cases} 1 \text{ if } s \text{ fits } f \\ 0 \text{ otherwise} \end{cases}$

1.2.3 Dilation

- Increases the size of structures
- Fills holes in regions
- $g(x,y) = \begin{cases} 1 \text{ if } s \text{ hits } f \\ 0 \text{ otherwise} \end{cases}$

1.2.4 Opening

- **Erosion** followed by **dilation**
- Removes small objects, preserves rough shape
- Filters out structures depending on the size and shape of the structuring element

1.2.5 Closing

- **Dilation** followed by **erosion**
- Fills holes, preserves rough shape

1.3 Labelling regions

1.3.1 Sequential connected components

- Process image from left to right, from top to bottom:
  1. If the current pixel value is 1
     (a) If only one neighbor **(left or top)** is 1, copy its label
     (b) If both neighbors are 1 and have the same label, copy the label
     (c) If they have different labels:
     – Copy label from the left
     – Update the table of equivalent labels (remember that the label of the left neighbor represents the same connected component as the label of the top neighbor)
     (d) Otherwise form a new label
- Relabel with the smallest equivalent labels

1.4 Color

- Additive models
- Subtractive models

1.4.1 Color spaces

- Role: unique color specification
- A color space is defined by the choice of primary colors (primaries)

1.4.2 Non-uniform color spaces

- If two colors are close to each other (by Euclidean distance), it **doesn't mean that they are similar perceptually**
- CIE XYZ — linear
- RGB — linear
- HSV — nonlinear

1.4.3 Uniform color spaces

- CIE Lab
- CIE u'v'

1.5 Color description by using histograms

- Histograms record the frequency of intensity levels
  $h(i) =$ the number of pixels in I with the intensity value i
- Color histogram is a robust representation of images (robust to changes in translation, scale and partial occlusion)

1.6 Filtering

- Noise reduction and image restoration
- Structure extraction/enhancement
- Filters are **separable** if they can be written as a product of two more simple filters

1.6.1 Types of noise

- Gaussian noise
- Salt and pepper noise
- Impulse noise

When chaining **linear** filters, the order **doesn't matter**. When chaining **nonlinear** filters (e.g. the median filter), the order **does matter**.

1.7 Linear filtering as template matching

- If we correlate the image with a template, we get a map of the template's similarity to the image
- The max value of the map is the location of the template in the image
- To account for the scale dissimilarities, we can start with the original version of our image, correlate with the template, scale the image **down** (subsample) and repeat the process — **image pyramid**
- **Smoothing** the image before subsampling removes the features that couldn't be reconstructed in the subsampled image because of **antialiasing**

2 Derivatives and edge detection

2.1 Image derivatives

Implementation by convolution:

- **Horizontal derivative:**
  - $\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$
  - kernel: $\begin{bmatrix} -1 & 1 \end{bmatrix}$
- **Vertical derivative:**
  - $\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1)-f(x,y)}{1}$
  - kernel: $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
- **Image gradient:**
  - $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$
  - It points in the direction of greatest intensity change
  - Gradient direction: $\theta = \arctan\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$
  - Gradient strength (magnitude): $\sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$

Because derivation amplifies the noise in the image, we usually smooth the image using a **2D Gaussian filter**:

- $I \to I * (\frac{\partial G}{\partial x})$
- The kernel size determines which structures are preserved: **bigger** Gaussian kernels detect edges on a **bigger** scale.

2.2 Edge detection

The basic idea: find **strong gradients**, then post-process
Criteria for an **optimal edge detector**:

- **Good detection:** minimizing the probability of false positives and false negatives
- **Good localization:** detecting edges close to their true location
- **Specificity:** minimizing the number of local maxima around the true edge (returning a single point per true edge)

2.2.1 Canny edge detector

1. Filter image by a **derivative of a Gaussian**
2. Calculate the gradient **magnitude** and **orientation**
3. **Thin** potential edges to a single pixel thickness
   Non-maxima suppression — for each pixel, check if it is a local maximum along its gradient direction. If it is not, set it to zero in the modified image.
4. Select sequences of connected pixels that are likely an edge
   Hysteresis — choose two thresholds, t_{low} and t_{high}. Discard all pixels below t_{low} and retain pixels that are above t_{high} or connected into a component that contains at least one pixel above t_{high}.

2.3 Edge detection by parametric models

2.3.1 Hough Transform

1. For each edge point compute parameters of all possible lines passing through that point
$$x \cos \vartheta + y \sin \vartheta = \rho$$
2. Cast a vote for each set of parameters (e.g. store votes inside an accumulator array)
3. Select the lines that recieve enough votes (i.e. elements of the accumulator array with the highest/high enough values)

Extensions:

- Use the gradient direction
  ϑ = gradient direction at (x, y)
- Assign higher weight (in votes) to points with large edge magnitude (instead of $H[\rho, \vartheta]$ += 1 use $H[\rho, \vartheta]$ += $m(x, y)$)

Pros:

- Each point is processed independently:
  - Robustness to partial occlusion
  - Highly parallelizable
- Robustness to noise
- Can detect multiple instances of a single model in one pass

Cons:

- Time complexity increases exponentially with the number of free parameters

3 Fitting parametric models

$$\mathbf{x}'_i = f(\mathbf{x}_i; \mathbf{p})$$
$$\mathbf{x}'_i = \mathbf{R}\mathbf{x}_i + \mathbf{T}$$

- Example: transform $\mathbf{x}_i$ into $\mathbf{x'}_i$ by a function $f(\mathbf{x}; \mathbf{p})$
- Inverse problem: **find the transformation parameters** based on a set of correspondences — best values for $\mathbf{p}$
- Best parameter values: those that **minimize** the **projection error**

TODO least squares

3.1 RANSAC

(**Ran**dom **Sa**mple **C**onsensus)

1. Randomly select the smallest group of correspondences from which we can estimate the parameters
2. Fit the parametric model to the selected correspondences (e.g. by the least squares method)
3. Project all other points and count the number of inliers
4. Remember the parameters that maximize the number of inliers

Parameters:

- **Threshold** for identifying the inliers (chosen in a way that we achieve the desired probability of an inlier falling below the threshold)

$$p_{fail} = (1 - w^n)^k$$
$$k = \frac{\log(p_{fail})}{\log(1 - w^n)}$$

- The number of **sampling iterations**

We typically need as many correspondences as there are parameters for our model.

Pros:

- Very simple and general
- Applicable to many real-life problems
- Often used in practice

Cons:

- We need to decide on the parameters
- Finding the optimum may require many iterations
- Fails at a very small number of inliers

4 Keypoints and matching

1. Keypoint **DETECTION**
2. Keypoint **DESCRIPTION**
3. Keypoint **MATCHING**

4.1 Single scale keypoint detection

- Harris corner detector
- Hessian corner detector

Both are rotation invariant, but **not invariant to scale change**.

4.1.1 Harris corner detector

$$M = \begin{bmatrix} \sum_{x,y} w(x,y) I_x(x,y) I_x(x,y) & \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) & \sum_{x,y} w(x,y) I_y(x,y) I_y(x,y) \end{bmatrix}$$

$$\Downarrow$$

$$M = \begin{bmatrix} G(\sigma) * I_x^2 & G(\sigma) * I_x I_y \\ G(\sigma) * I_x I_y & G(\sigma) * I_y^2 \end{bmatrix}$$

4.1.2 Hessian corner detector

- Determinant of a Hessian

$$\text{Hessian}(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$

$$\det(\text{Hessian}(I))$$

4.2 Scale selection

- Blob detection

4.2.1 Laplacian of Gaussian (LoG)

$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

$$L = \sigma^2 (G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma))$$

- Laplacian pyramid
- LoG can be well approximated with a difference of Gaussians

4.2.2 Difference of Gaussians (DoG)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

- this does not require computation of second derivatives

4.3 Local descriptors

- How to describe keypoints?

4.3.1 Scale Invariant Feature Transform (SIFT)

1. Split region into 4×4 sub-regions: 16 cells
2. Calculate **gradients** on each pixel and **smooth** over a few neighbours
3. In each cell calculate a **histogram** of gradient orientations (8 directions)
4. Descriptor (stack histograms into a vector and normalize): 4×4×8 = 128 dimensions

SIFT **is not** rotation invariant, but it is fast and very robust to intensity changes.

4.3.2 Finding correspondences using keypoints

Strategies:

1. For each keypoint in the left image identify the most similar keypoint in the right image
2. Keep only symmetric matches
   - i.e. keep only correspondences between every two points A and B, where A is a point in the left image and B is its match in the right image, so that B is most similar to A among all points in the right image and vice versa.
3. Calculate the distances between:
   - point A and the **second most similar** keypoint in the right image
   - point A and the **most similar** keypoint in the right image

   and then calculate their ratio
   - it will be **low for distinctive** keypoints and **high for distinctive** keypoints

5 Camera geometry

Two kinds of projections

1. **Extrinsic** projection: 3D World → 3D Camera
2. **Intrinsic** projection: 3D Camera → 2D Image

5.1 Calibration matrix

Prescribes the projection of a 3D point in camera coordinate system into pixels.

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\alpha_{\mathbf{x}}$, $\alpha_{\mathbf{y}}$ — scale
$\mathbf{x_0}$, $\mathbf{y_0}$ — translation
$\mathbf{s}$ — skewing factor

World-to-camera coordinate system transformation:

$$\tilde{X}_{cam} = R(\tilde{X} - \tilde{C})$$

$\mathbf{R}$ — the camera's rotation matrix
$\tilde{\mathbf{C}}$ — camera origin in world c. s.
$\tilde{\mathbf{X}}$ — a 3D point in the world c. s.
$\tilde{\mathbf{X}}_{\mathbf{cam}}$ — the same point, represented in the camera c. s.

5.2 Camera parameters

5.2.1 Intrinsic

Parameter	Degrees of freedom
Principal point coordinates	2
Focal length	1
Pixel scaling factor (rectangular pixels)	1
Shear (non-rectangular sensor array)	1
Radial distortion	

5.2.2 Extrinsic

Parameter	Degrees of freedom
Rotation	3
Translation	3

5.3 Vanishing points

- Sets of 3D parallel lines intersect at a vanishing point
- Horizon is a collection of all the vanishing points corresponding to **a set of parallel planes**

5.4 Camera calibration

- Estimating the projection matrix from a known calibration object
- Proper calibration reqpires measuring the points at a sub-pixel accuracy
- Highly depends on the calibration pattern
- A rule of thumb:
  - Number of constraints should exceed the number of unknowns by a factor of 5
  - e.g. for **11 parameters** in P, use **at least 28 points** (2 equations per point pair)

6 Multiple-view geometry

6.1 Stereo geometry and scene reconstruction

6.1.1 Epipolar geometry

- **Baseline:** a line connecting the camera centers
- **Epipole:** a point where the baseline punctures the image plane
- **Epipolar plane:** a plane connecting two epipoles and a 3D point
- **Epipolar line:** an intersection of the epipolar plane and an image plane

6.1.2 Essential matrix

- Denoted by $\mathbf{E}$
- A 3D point is mapped to points $\mathbf{p}$ and $\mathbf{p}'$ which are related by $\mathbf{p^T E p'} = \mathbf{0}$

6.2 Structure from motion (SFM)

6.2.1 Fundamental matrix

- Denoted by $\mathbf{F}$
- $F = K^{-T} E K'^{-1}$

6.2.2 Fundamental matrix estimation

1. Find keypoints in each image
2. Calculate possible matches
3. Robustly estimate the epipolar geometry by RANSAC

6.2.3 Normalized 8-point algorithm

1. Precondition: center image points and scale such that the standard deviation becomes $\sqrt{2}$ pixels
2. Apply 8-point algorithm to calculate $\tilde{\mathbf{F}}$ from the preconditioned points
3. Enforce $\mathbf{rank} = \mathbf{2}$
4. Transform the fundamental matrix back to original units

6.3 Active stereo

- Uses **one camera** and a **projector** (e.g. a laser pointer)

7 Recognition

How to come up with features?

1. Natural (linear) coordinate systems:
   - For some applications it is enough just to linearly transform the input data
   - **PCA/LDA**
2. Handcrafted nonlinear transforms:
   - Nonlinear transforms improve feature robustness
   - **HoG**
3. Feature selection:
   - Machine learning to select optimal features from a pool of several handcrafted transforms
   - **AdaBoost**
4. End-to-end learning of feature transform:
   - Have a machine learn the entire feature extraction and selection pipeline

7.1 Learning natural coordinate systems by subspace methods

An image of a face can be treated as a high-dimensional gray-level vector ($100\times100 = 10{,}000\times1$ vector)

7.2 Reconstruction subspace: Principal component analysis (PCA)

M — the number of pixels in the image
N — the number of training examples (typically much lower than M)

1. Estimate the mean vector: $\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$
2. Center the input data around the mean: $\hat{\mathbf{X}}_i = \mathbf{X}_i - \mu$
3. if $M \leq N$ then:
   (a) Estimate the covariance matrix: $\mathbf{C} = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^{\mathbf{T}}$
   (b) Perform SVD on $\mathbf{C}$. Obtain eigenvectors $\mathbf{U}$ and eigenvalues λ.
4. else:

(a) Estimate the inner product matrix: $\mathbf{C}' = \frac{1}{N}\hat{\mathbf{X}}^{\mathbf{T}}\hat{\mathbf{X}}$

(b) Perform SVD on $\mathbf{C}'$. Obtain eigenvectors $\mathbf{U}'$ and eigenvalues λ'.

(c) Determine the eigenvectors $\mathbf{U}$: $\mathbf{u}_i = \frac{\hat{\mathbf{X}}\mathbf{u}_i'}{\sqrt{N\lambda_i'}}, i = 1 \dots N$

(d) Determine the eigenvalues $\lambda = \lambda'$

5. end if

- Even in PCA, the same object, viewed under different conditions (illumination, orientation) projects into different parts of the subspace. This means that we could generate different appearances by accessing specific parts of the subspace.
- Linear encoders (PCA) are quite weak for storing various kinds of appearances — **Neural-network encoders are more powerful**!

7.3 Discrimination subspace: linear discriminant analysis (LDA)

Find a subspace that:

- **Maximizes** distances between classes
- **Minimizes** distances within classes

7.4 Handcrafted non-linear transforms

7.4.1 HoG

- Histogram of gradients

7.5 Learning features by feature selection

To apply in real-time applications:

1. Feature extraction should be fast
2. Classifier application should be fast

7.5.1 AdaBoost

- Builds a strong classifier which is a weighted sum of many weak classifiers
- Simple to implement

1. Train a sequence of weak classifiers.
2. Each weak classifier splits training examples with at least 50% accuracy
3. incorrectly classified examples get more weight in training the next weak classifier

7.5.2 Cascade of classifiers

When going over the image with a sliding window, we can save time by applying only a first few classifiers to reject the windows that very likely do not contain the particular category. The "surviving" regions are then re-classified with stronger classifiers.

7.6 End-to-end feature (and classifier) learning

7.6.1 Convolutional neural networks (CNN)

- Convolutional layers
- Non-linearity (RELU)
- Pooling layers

Convolutional layers — they convolve the input and pass the result to the next layer

- Input: e.g. an $32 \times 32 \times 3$ image, N filters
- Output: a $32 \times 32 \times N$ tensor

Nonlinear layers — e.g. Rectified linear unit (RELU), implement nonlinear feature transformations (e.g. setting negative values to 0)

- Input: N inputs
- Output: N outputs

Pooling layers — aim to increase the receptive field without significantly increasing the number of parameters