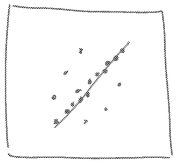
IIT cs512 Computer Vision – Fall 2024 ©

Model Fitting

We have a model which we wanted to fit and find the values of the parameters

Line detection

Tasks:
1) Grouping
2) Fitting



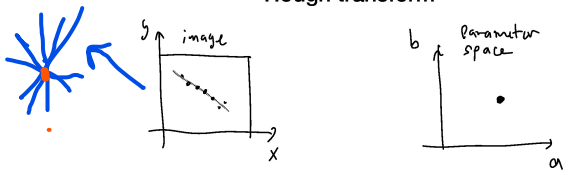
* Two simultaneous tasks. solving one makes the other easier

Grouping - group the points that are closer to each other
Fitting - Fit the line that best captures the points

If one of the above tasks is solved, the other is easier.

Hough transform

Works well for lines alone, not for others



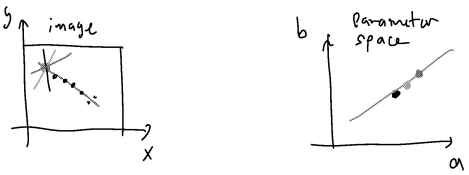
$y = ax + b$

* multiple points on a line in the image correspond to a single point in parameter space (the parameters of the line)

* Detect lines by casting votes in parameter space

Each point on the image, XY plot (eg-red point) will vote for the paramaters of all the lines that it could possibly belong to. The votes are plotted in the parameter space ab.

Voting in parameter space



* A single point in the image defines a line in parameter space (describing the parameters of all possible lines through the point)

$y = ax + b \rightarrow b = y - ax$

given (x,y) scan a and compute b.
cast votes at locations (a, b) - a line

in XY plot, x is the slope and y is the intercept
in ab plot, a is the slope and b is the intercept

using formula, for every point(x,y)...we calculate b for a range of 'a' values--->a point in image is represented as a line in parameter space.

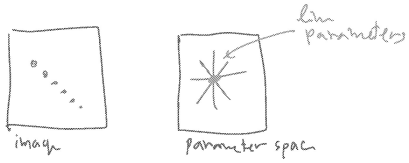
similarly, it is done for all points in the image which brings in lines in the parameter space. All the lines in the ab space will meet at a single point which represent the parameter (a,b) of the fitting line

Refer images in Notes

How do we decide the range of 'a' values that we would scan for?? what is a?--it is the slope of the fitting line we are trying to find. The line we try to find is y =ax+b,

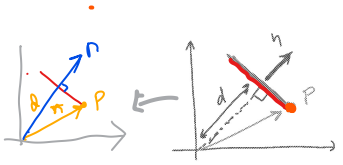
Voting in parameter space

- Each point in the image defines a line in parameter space
- Multiple points define multiple lines in parameter space
- Intersection of lines in parameter space indicate parameters common to multiple points on the same line



Better line representation

* Problem with $y = ax + b$ model:
  - what is the possible range of a?
  - How to represent vertical lines



$n = (n_x, n_y)$ normal (orientation)

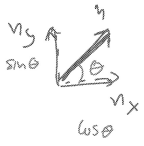
d = distance from origin

* For point $P = (x, y)$ to be on the line:

$(n_x, n_y) \cdot (x, y) = d \Rightarrow n_x(x) + n_y(y) - d = 0$

Implicit line equation

$n = (n_x, n_y)$ $\quad \begin{cases} n_x = \cos\theta \\ n_y = \sin\theta \end{cases}$



Assume n to be a unit vector, n=1

* Implicit line equation:

$(n_x, n_y) \cdot (x, y) = d \Rightarrow (\cos\theta)x + (\sin\theta)y - d = 0$

3 parameters $\quad$ $(Ax + By + C = 0)$

2 parameters: θ, d

* line parameters: θ, d

Since, the slope 'a' goes to infinity in the above definition of line y=ax+b, it is not possible to find the line accurately.

So, we move on to a different approach (vector) representing a line.

What do we need to find a line--->
- its orientation - found with the help of the normal vector to the line
- its distance from origin (magnitude) -->how do we find this?

Say, the red line is the fitting line we are trying to find. We project the every point(P) on the line to the normal vector which would give the distance d.

projection is given by dot product, p.n = d --->Implicit line equation --> because we did not get the x,y parameters of the fitting line
The same formula holds for hyperplane

→ We have to estimate what nx and ny are

Hough with explicit line equation

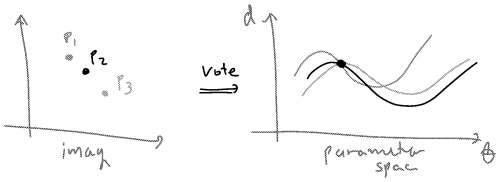
* Explicit line equation:

$x \cos\theta + y \sin\theta = d$

* Vote by point (x, y):

for each θ_i: Vote $d_i = x \cos\theta_i + y \sin\theta_i$

d is a sum of two curves-->d is a sinusoidal curve too



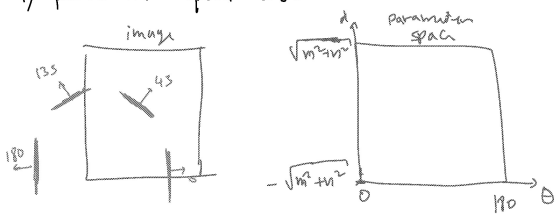
The point in parameter space where all the curves (that corresponds to the points in the image) meet represents the parameters of the fitting line

Refer image

Here, we dont worry about range of theta, it is finite and ranges from 0 to 360 degrees

Practical issues

1) parameter space size:



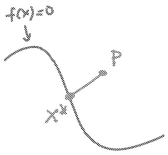
2) Bin size: larger bins are more efficient but provide less localization

3) Peak detection: threshold and suppress close peaks

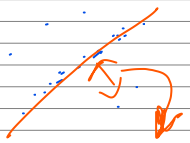
Geometric distance

* Distance from a point to arbitrary implicit curve: $f(x) = 0$

$$\begin{cases} d(p, f) = |p - x^*| \\ x^* \text{ is closest point} \end{cases}$$



xcos theta + y sin theta -d = 0 .....the 0 on right says that we are trying to minimize the distance between the points and the fitting line. it is called the algebraic distance



The algebraic distance is the same as the geometric distance in case of lines alone

Geometric distance is valid/has meaning only if its a geometric object. Fitting a line has geometric distance but parameters of a camera dont-as they dont have a geometric meaning

* solution:

$$d(p, f) = |p - x^*| = \frac{|f(p)|}{|\nabla f(x^*)|}$$

$|f(p)|$ ← algebraic distance

geometric distance

$|\nabla f(x^*)|$ → gradient

Let the curve be f(x,y) = xcos theta+y sin theta
f(xy) = 0 -->points on the curve.
for an arbitrary curve, there is no guarantee that f(x) calculation would give the distance of that point from the curve. we know that f (x) = 0 is the algebraic distance but we dont if its the geomteric distance a well. so, we have to find the closest point on the curve to our point in question P. How to find the geametric distance--->proved solution-->

Refer images

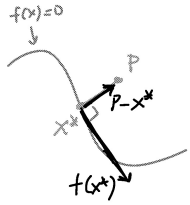
but how to find the closest point???? we have to solve as below.
X* has 2 properties if its the closest point on the curve.
f(X*)=0 and the vector joining X* to point P should be perpendicular to the tangent of the curve.

Geometric distance

* To find x^* solve:

$$\begin{cases} f(x^*) = 0 \\ (p - x^*) \cdot t(x^*) = 0 \end{cases}$$

($t(x^*)$: tangent at x^*)



* To find tangent at x^*
  1) compute gradient: $\nabla f(x^*) = \left[\frac{\partial f}{\partial x} \; \frac{\partial f}{\partial y}\right]^T$
  2) Rotate by 90°: $t = R(90)\nabla f(x^*) = \left[\frac{\partial f}{\partial y} \; -\frac{\partial f}{\partial x}\right]^T$

gradient at X* is the normal vector-->rotate by 90 ->we get the tangent

This approach seems fine but it is not practical to find the optimal X* by this optimization algorthm

Geometric distance

* Approximated solution:

$$d(p,f) = |p - x^*| = \frac{|f(p)|}{|\nabla f(x^*)|} \approx \frac{|f(p)|}{|\nabla f(p)|}$$

(geometric distance ≈ algebraic distance)

⇒ reduce the algebraic distance for points with large gradient $|\nabla f(p)|$

we approximate and take the gradient of p instead of x* because if they are close enough , the gradient would be same

Geometric distance

* Geometric distance fitting:

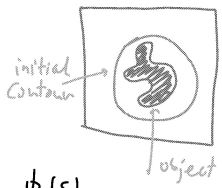
$$\begin{cases} E(\ell) = \sum_i \frac{|f(p_i;\ell)|}{|\nabla f(p_i;\ell)|} \\ \text{where} \quad f(p_i;\ell) = \ell^T p_i \\ \ell^* = \arg\min_\ell E(\ell) \quad \text{s.t.} \quad b^2 - 4ac < 0 \end{cases}$$

was not taught

* No explicit solution to $\nabla E(\ell) = 0$
  ⇒ iterative numerical solution
  (e.g. Gradient descent)

Active contours (snakes)

- Gradually deform initial contour to fit object boundaries



- Use parametric curve $\phi(s)$ to represent contour

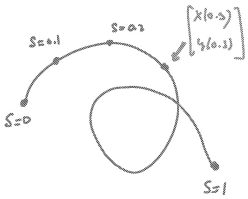
For a curve/line, we can write parameters, but for arbitrary shape, we cant.

So, we fit the image to a contour by iteratively trying to find the fit for the image

phi is the coordinates of the points on the contour and s is a scalar
How to find the phi??

Parametric curves

$$\phi(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$$

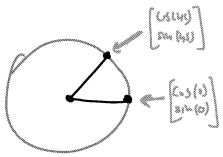


If we can define the functions x(s) and y(s), then we will be able to get the contour based on the range of s values

The function of the x and y can be any polynomial

* example: circle

$$\phi(s) = \begin{bmatrix} \cos(s) \\ \sin(s) \end{bmatrix}$$



Error functional

The unknown we are seeking is a function $\phi(s)$

$\Rightarrow$ error functional

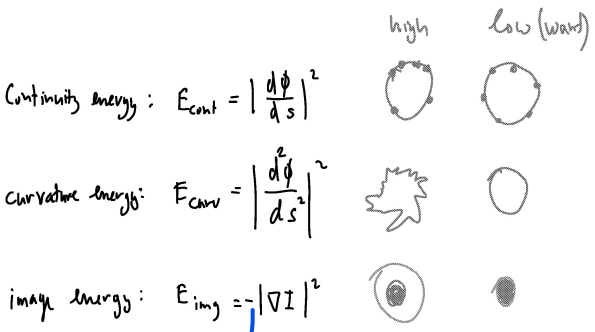
$$E[\phi(s)] = \int_{\phi(s)} \underbrace{\alpha(s)E_{cont} + \beta(s)E_{curv}}_{\text{internal energy}} + \underbrace{\gamma(s)E_{img}}_{\text{external energy}} \, ds$$

$\alpha(s)$, $\beta(s)$, $\gamma(s)$ are coefficients of the different energy terms

we need low continuous (high energy) , low curvature(mroe smooth) and low image energy(well fits the image)

high curvature may indicate overfitting

Error functional



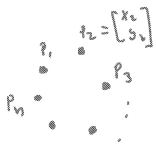
disintegrate the original contour to points and then adjust the points to fit the image--->continuous to discrete

so that we can E as a minimizing function because when gradien value is high, it is good, we say -ve to show high is undesired

Robust estimation

Discrete functional

Discrete case: $\phi(s) \rightarrow \{p_i\}_{i=1}^n$



$$E_{cont} = \left|\frac{d\phi}{ds}\right|^2 = |p_{i+1} - p_i|^2$$

$$E_{curv} = \left|\frac{d^2\phi}{ds^2}\right|^2 = |(p_{i+1} - p_i) - (p_i - p_{i-1})|^2$$
$$= |p_{i+1} - 2p_i + p_{i-1}|^2$$

$$E_{img} = -|\nabla I|^2$$

Discrete functional

$$E(\{p_i\}) = \sum_{i=1}^{n} \alpha_i \left(|p_{i+1} - p_i| - d\right)^2 + \sum_{i=1}^{n} \beta_i |p_{i+1} - 2p_i + p_{i-1}|^2 - \sum_{i=1}^{n} \gamma_i |\nabla I(p_i)|^2$$

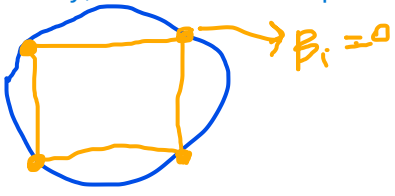
average distance between points to prevent shrinking

$\alpha_i, \beta_i, \gamma_i$ are user selected parameters
$\{p_i\}$ are the unknown model parameters
d is a computed parameter
(also zero β_i at corners to allow discontinuity)

prevent shrinking all the points of the contour to a single point
so, we have to make sure the points are evenly spaced

There are 2 ways to solve-
1. we can either find the explicit derivatives and equate to zero
2. numerical solution - we can guess the initial value of s and then minimize the loss function by computing the gradient and guiding it to the minimal

we dont use the same values of alpha,beta, gamma throughut we want continuity,curvature in some point and not in some places like the



Discrete functional

To minimize the error functional E:

- start with initial guess $\{p_i\}_{i=1}^n$
- compute d (average distance between points)
- For each point p_i try to minimize E by testing what happens when moving to neighboring locations
- Repeat while E is decreasing

Selecting coefficients

- Select $\alpha_i, \beta_i, \gamma_i$ to normalize the different terms to a similar scale and set the energy terms importance
- To allow for piecewise curves, set $\beta_i = 0$ to points with high curvature:
  i.e. when: $|p_{i+1} - 2p_i + p_{i-1}| > \tau$

Refer images in Notes

the gradient terms associated with alpha, beta, gamma may be higher and of different scale, so we can the coefficients (a,b,g) to normalize the term

Marking of corners--->it means it has high curvature. therefore, when we keep calculating the curvature, if we find the value of curvature to be greater than a threshold (hyper parameter), then we set beta(i) as 0 allowing discontinuity of points so that it can be a corner

piecewise curve--->we actually build the curve from pieces/points. The points are generally continuous, but some points are not continuous which can be found with the help of curvature values