Vision and Image Processing: Segmentation – Snakes

François Lauze
Department of Computer Science
University of Copenhagen

Plan for today and Wednesday

- Discuss a general introduction of segmentation.
- Present some standard “old-fashioned” techniques.
- Introduce the notion of Active Contours.
- Discuss the algorithm of Kass, Witkin and Terzopoulos.

A “disclaimer”: for those who followed Signal and Image Processing, the first part of the material presented here is rather similar to the segmentation lecture of SIP, but don’t worry, the second part will be more interesting :-)

Outline

Image Segmentation

- An intelligible image is not formed of random pixels. There must fundamental consistencies between them.
- Image segmentation is the process of dividing an image into coherent regions of similarity, called segments, by grouping similar pixels:
  - Region: a group of connected pixels that share some common properties.
  - Property: intensity, color, texture, motion (for sequences), boundary (edges)
  - Some of these properties can be defined in a pixelwise manner: intensity, color for instance. Some are related to pixel neighborhoods, texture for instance.

- Two different segments should have dissimilar properties. In particular, boundary between segments should present large variations.

- Two main approaches:

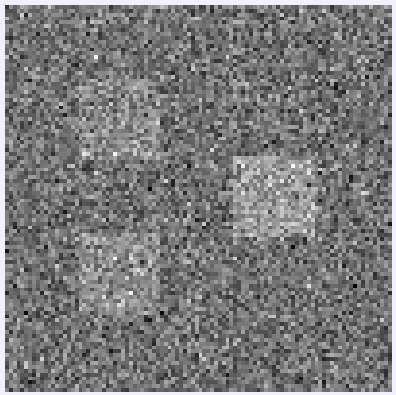
  - Region based segmentation.

  - Edge / contour based segmentations.

Regions and Edges

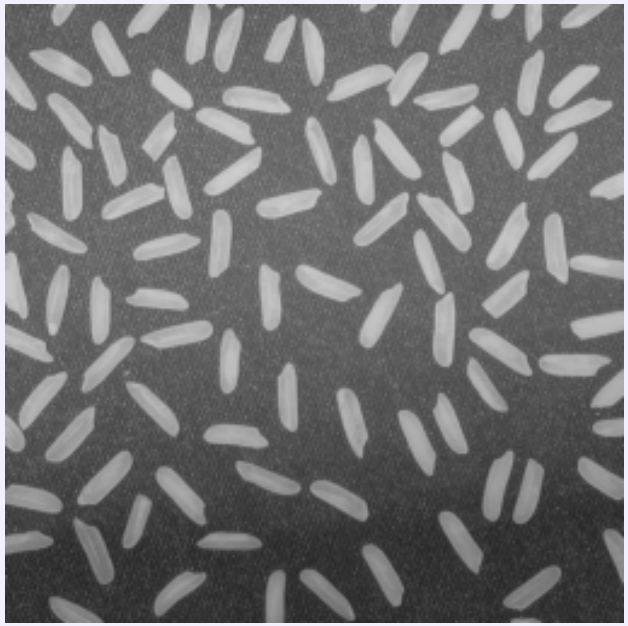
- A region should be bounded by a closed contour: edge detection.
- Regions may be obtained by “boundary filling”.

- However edges are not always well defined in observed images.



Regions

- Regions correspond generally to objects or pieces of objects in a scene.
- Scenes may contain several objects.



Outline

Edge Based Segmentation

- Boundary between region corresponds to sharp intensity / color / texture change.
- Derivatives should indicate this:

- Edges should correspond to local maximum of gradient magnitude.
- Gradient of f : $\nabla f = (f_x, f_y)^T$, Gradient magnitude = $\sqrt{f_x^2 + f_y^2}$.
- Zero-crossing of Laplacian $\Delta f = \nabla^2 f := f_{xx} + f_{yy}$ is also an indicator.
- How to compute image derivatives / gradient?

Image Derivatives - First Order

- Finite difference approximation:
  - Forward differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+h,y)-f(x,y)}{h}, \quad \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+h)-f(x,y)}{h}$$

  - Backward differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x,y)-f(x-h,y)}{h}, \quad \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y)-f(x,y-h)}{h}$$

  - Central differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+h,y)-f(x-h,y)}{2h}, \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+h)-f(x,y-h)}{2h}$$

- For those familiar with it, they can be implemented by convolution!

Second Order Derivatives

- xx-direction

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2},$$

- yy-direction

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{f(x,y+h) - 2f(x,y) + f(x,y-h)}{h^2},$$

- xy-direction

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} \approx \frac{1}{4h^2}\Big(- f(x-h,y+h) + f(x+h,y+h) + f(x-h,y-h) - f(x+h,y-h)\Big)$$

- Here too, convolution!

Derivatives and noise

- Derivation filters are high pass: they amplify high pass signals and noise is high pass.
- Solution: Apply smoothing filter prior to differentiation.

- Better: combine smoothing filter and differentiation!

- Remember Kim's lectures!

$$D(g * f) = Dg * f$$

Derivatives and Gaussian

- Convolution by Gaussian smooths: noise removal. Standard deviation of Gaussian directly linked to scale of features.
- Derivative of Gaussian:

$$g_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad \frac{\partial g_\sigma}{\partial x} = -\frac{x}{\sigma^2} g_\sigma, \quad \frac{\partial g_\sigma}{\partial y} = -\frac{y}{\sigma^2} g_\sigma$$

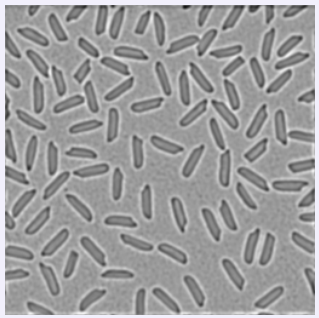
- Laplacian of Gaussian (LoG)

$$\Delta g_\sigma = \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2} = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} g_\sigma$$

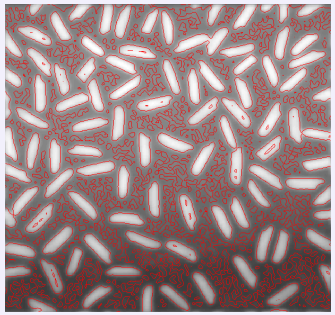
Marr-Hildreth Edge Detector

D. Marr and E. Hildreth, 1980.

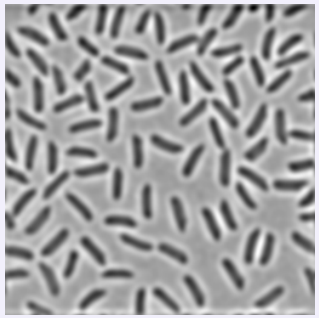
- Observe zero-crossing of Laplacian of Gaussian (the same use for SIFT for instance)
- Convolve image f with Laplacian of Gaussian filter: $\Delta_\sigma f = \Delta g_\sigma * f$
- Detect the zero-crossings of $\Delta_\sigma f$



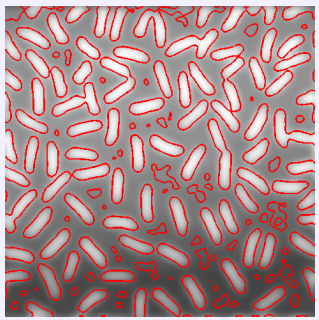
LoG $\sigma = 1.4$



detected edges



LoG $\sigma = 3.0$



detected edges

Canny Edge Detector

J. Canny, 1986.

- Develop an algorithm optimal wrt:
  - Detection: maximize probability of detection of true edges, minimize probability of detection of false ones.
  - Localization: detected edges as close as possible to real ones.
  - Number of responses: 1 real edge should not result in 2 or more detected ones.
- Canny algorithm attempts to answer optimally to these criteria.
- Divided in 5 steps.

Algorithm - I

1. Smoothing. Noise removal by Gaussian smoothing with 2D-Gaussian g_σ.
2. Gradient computation. Can be combined with step 1 by convolution with Gaussian derivatives. Gives $f_{x\sigma}$ and $f_{y\sigma}$. Potential edges marked as gradients with large magnitudes. Directions computed by

$$\theta = \arctan\left(\frac{|f_{y\sigma}|}{|f_{x\sigma}|}\right).$$

3. Non-maximum suppression.
   - Round gradient direction to nearest 45° (connectivity of a 8-points neighborhood).
   - Compare gradient magnitude of current pixel with magnitudes of neighbors in the gradient direction. That is, if $\theta = 90°$ compare with north and south pixel, if $\theta = 135°$ compare with ? (this is a question for you!)
   - If gradient magnitude of the current pixel is largest, keep it, otherwise suppress it (say set to 0).

Algorithm - II

1. Double thresholding. Non suppressed edges marked with their gradient magnitude. Some are true edges, some may be noise. Two thresholds τ_{high} and τ_{low} are used. Edges with magnitude $\leq \tau_{high}$ marked as strong edges. Edges with magnitude betweenτ_{low} and τ_{high} marked as weak edges. Remaining are suppressed.

2. Hysteresis edge tracking Strong edges are certain and included in the final image. Weak edges included only if connected (via other weak edges) to strong edges. Connectivity means existence of a chain of neighbor weak edges pixels from candidate edge to a strong edge.

Canny Example

$\tau_{low} = 0.08$, $\tau_{high} = 0.2$

source image

$\sigma = 1.4$

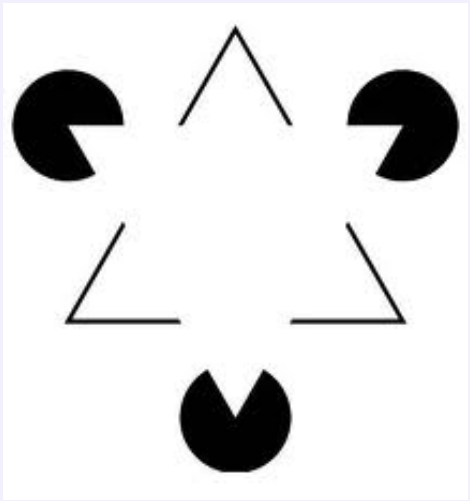
$\sigma = 3.0$

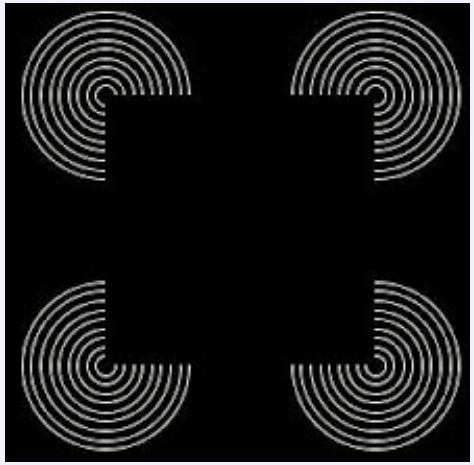
- Edges are not closed: good but might be insufficient for segmentation.

Outline

Contours and Perception

A Classical example from Italian psychologist G. Kanizsa[1]



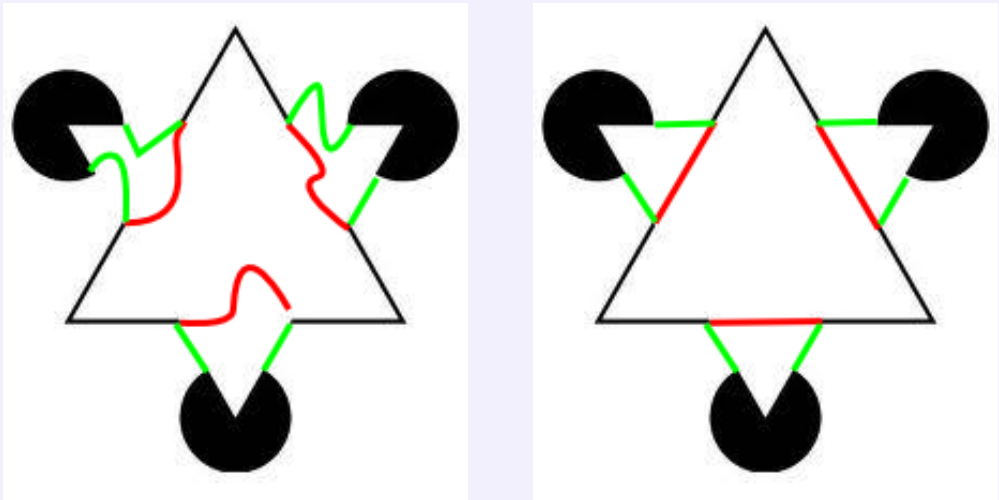


The brain is “closing the gaps”!

[1] Grammatica del vedere / Organization in Vision

But how...

What is the best continuation?



- We reconstruct the gap with an implicit assumption of simplicity / regularity for the contour.!
- Variational segmentation algorithms operationalize it!

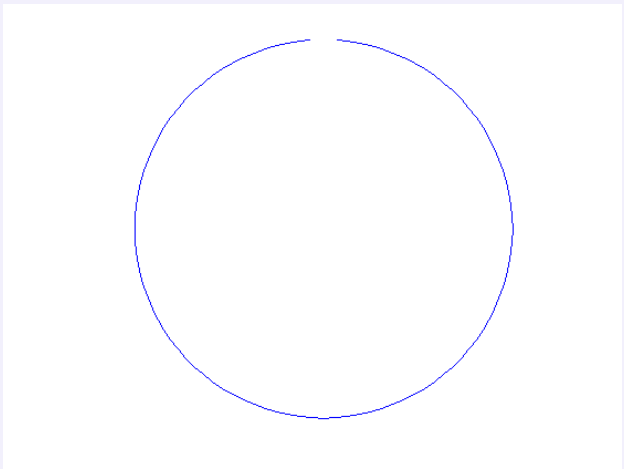
Contour representation

- The simplest way to represent a 2D contour is to use a curve.
- Mathematically, it is a function

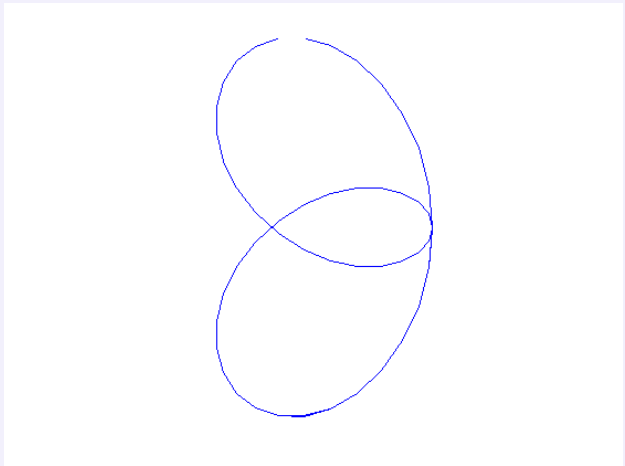
$$C : p \in [a, b] \mapsto C(p) = (x(p), y(p)) \in \mathbb{R}^2$$

- If $C(a) = C(b)$ the curve is closed.

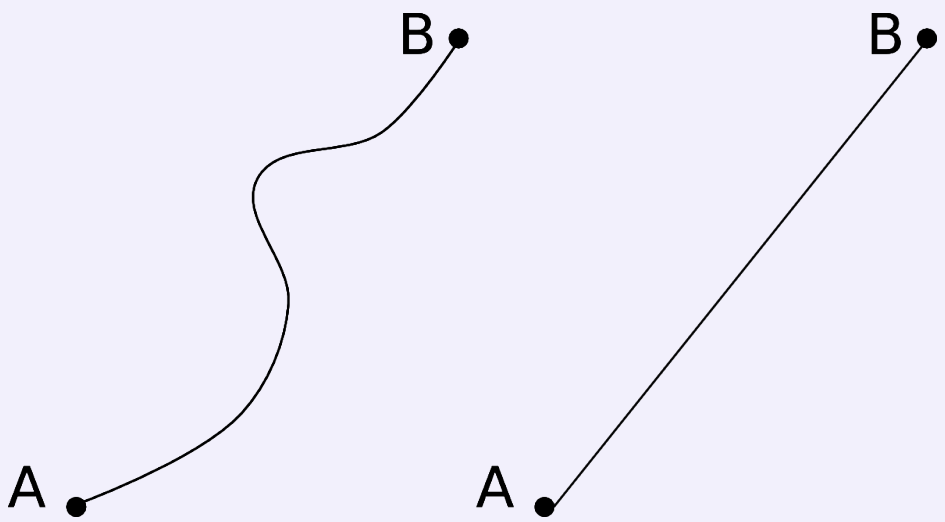
$t \mapsto (\cos(p), \sin(p)$



$p \mapsto (\cos(2p)^2, \sin(p)\cos(3p)$



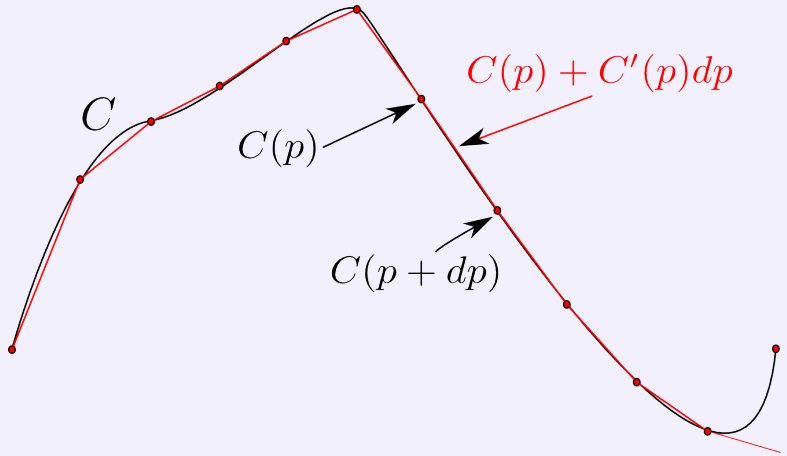
Simplicity / Regularity



- The first curve is longer and winds more than the second one.
- How can we put numbers on these ideas?

Curve length

This is computed via the *derivative* of the curve.



Line approximation between 2 points, length is $|C(p+dp)-C(p)|$ (|-| means norm of vectors in $\mathbb{R}^2$). But differential calculus says

$$C'(p) = \lim_{dp \to 0} \frac{C(p+dp)-C(p)}{dp}$$

So for small dp,

$$C'(p) \approx \frac{C(p+dp)-C(p)}{dp}, \quad C(p+dp)-C(p) \approx C'(p)dp.$$

Finally, One gets an approximation of the length by

$$\ell(C) \approx \sum_{i=1}^{k} |C'(p + idp)|dp$$

Making $dp \to 0$ one gets

$$\ell(C) = \int_a^b |C'(p)|dp$$

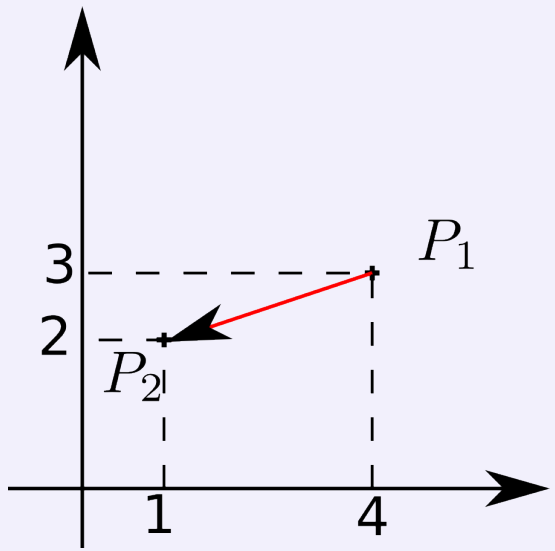
The length of C is a measure of complexity of the curve.
Related to the length is the curve energy

$$\mathcal{E}(C) = \int_a^b |C'(p)|^2 \, dp$$

Discrete Formulation

- In the sequel, subtracting two points in the plane means getting the vectors joining them, and means subtracting their coordinates

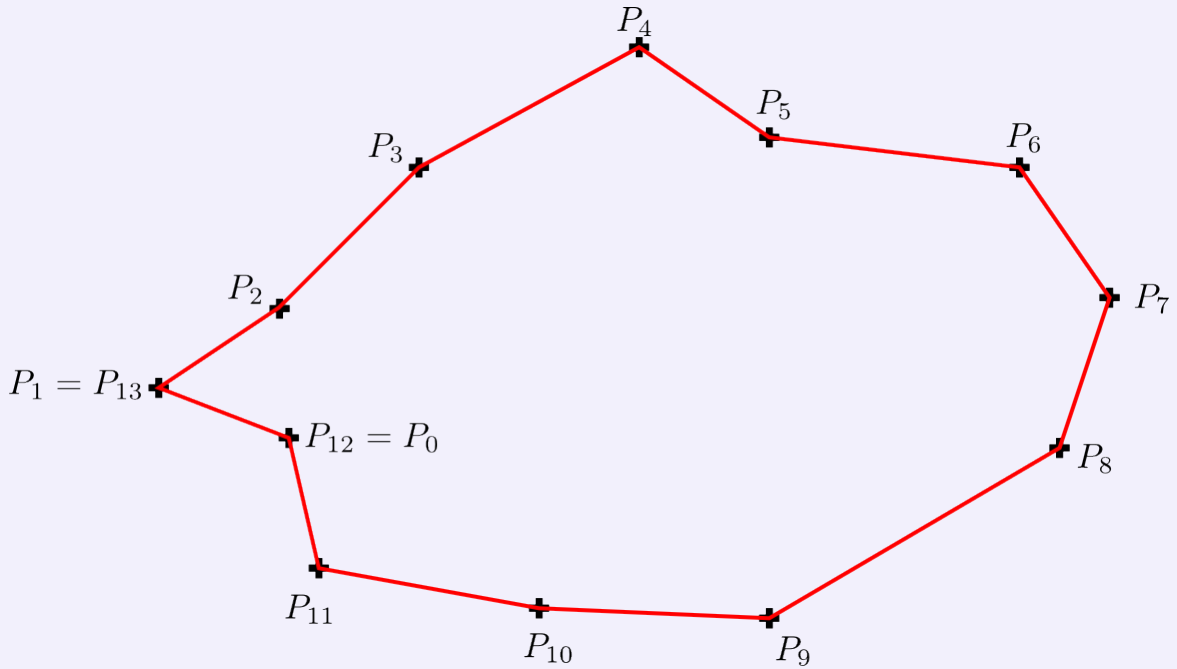
$$P_1 = (4,3), \quad P_2 = (1,2), \quad P_2 - P_1 = \overrightarrow{P_1 P_2} = (-3,-1).$$



- The length of vector $P_2 - P_1$ is denoted $\|P_2 - P_1\|$ and is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Curve representation

- A closed curve C will be represented by n points $P_1, P_2, \ldots, P_n$, and consecutive points are joined by line segments, and with $P_{n+1} = P_1,$, $P_0 = P_n$ indicating that the curve is closed:



A closed curve with 12 points

Discrete Snake Energy: Curve Internal Forces

- Curve Energy term: sum of squared length of the curve segments:

$$E_{\mathcal{C}}(C) = \sum_{i=1}^{n} \|P_{i+1} - P_i\|^2, \quad P_{n+1} = P_1.$$

- Bending energy term: sum squared-length of differences of consecutive segments

$$\begin{aligned} E_{\mathcal{B}}(C) &= \sum_{i=1}^{n} \|(P_{i+1} - P_i) - (P_i - P_{i-1})\|^2 \\ &= \sum_{i=1}^{n} \|P_{i+1} - 2P_i + P_{i-1}\|^2, \quad P_{n+1} = P_1, P_0 = P_n. \end{aligned}$$

Discrete Snake Energy: External (Image) Forces

- F is an image derived from the image I to be segmented, with the property that F should be small when at an edge of I and large when at a flat region of I.
- Standard example is

$$F = -\|\nabla I_\sigma\|^2 = -\left(I_{\sigma x}^2 + I_{\sigma y}^2\right)^2.$$

  with $I_\sigma = g_\sigma * I$, I convolved with Gaussian of standard deviation σ.
- External force at point $P = (x, y)$: $F(P) = F(x, y)$ and external / image energy:

$$E_{\mathcal{E}}(C) = \sum_{i=1}^{n} F(P_i).$$

External Forces: Example

- Using the force term $F = -\|\nabla I_\sigma\|^2$, dark indicates edge:

Original image

Force term with $\sigma = 3$

Snake Energy

- Given a discrete closed curve $C = (P_1, P_2, \ldots, P_n)$, its Snake Energy is

$$\begin{aligned} E_{\mathcal{S}}(C) &= \alpha E_{\mathcal{C}}(C) + \beta E_{\mathcal{B}}(C) + \gamma E_{\mathcal{E}}(C) \\ &= \alpha \sum_{i=1}^{n} \|P_{i+1} - P_i\|^2 + \beta \sum_{i=1}^{n} \|P_{i+1} - 2P_i + p_{i-1}\|^2 + \gamma \sum_{i=1}^{n} F(P_i) \\ &= \sum_{i=1}^{n} \left(\alpha \|P_{i+1} - P_i\|^2 + \beta \|P_{i+1} - 2P_i + P_{i-1}\|^2 + \gamma F(P_i)\right). \end{aligned}$$

- α, β and γ are 3 weights (i.e., positive numbers) that provide the trade-off between the different parts.

With Coordinates

- $C = (P_1, \ldots, P_n) = ((x_1, y_1), \ldots, (x_n, y_n)) \rightsquigarrow (x_1, \ldots, x_n, y_1 \ldots, y_n)$
- $E_{\mathcal{S}}(C) = E_{\mathcal{S}}(x_1, \ldots, x_n, y_1, \ldots, y_n)$

$$E_{\mathcal{S}}(C) = \alpha \sum_{i=1}^{n} \left((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2\right) +$$

$$\beta \sum_{i=1}^{n} \left((x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2\right)$$

We assume that $x_{i+1} = x_1$, $y_{i+1} = y_1$ (closed curve).

- $E_{\mathcal{E}}(C) = E_{\mathcal{E}}(x_1, \ldots, x_n, y_1, \ldots, y_n)$

$$E_{\mathcal{E}}(C) = \sum_{i=1}^{n} F(x_i, y_i)$$

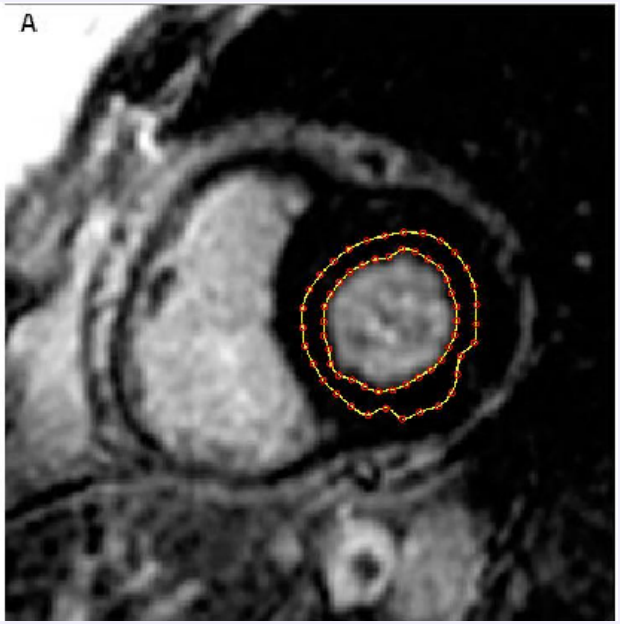
Gradient

- $\nabla_C \mathcal{E} = \left(\frac{\partial E}{\partial x_1}, \ldots, \frac{\partial E}{\partial x_n}, \frac{\partial E}{\partial y_1}, \ldots, \frac{\partial E}{\partial y_n} \right)^T$
- Smoothness term:

$$\frac{\partial E_S}{\partial x_i} = -2\alpha \left(x_{i+1} - 2x_i + x_{i-1} \right) + \ldots$$

- remaining computations left as an exercise!

Snake example

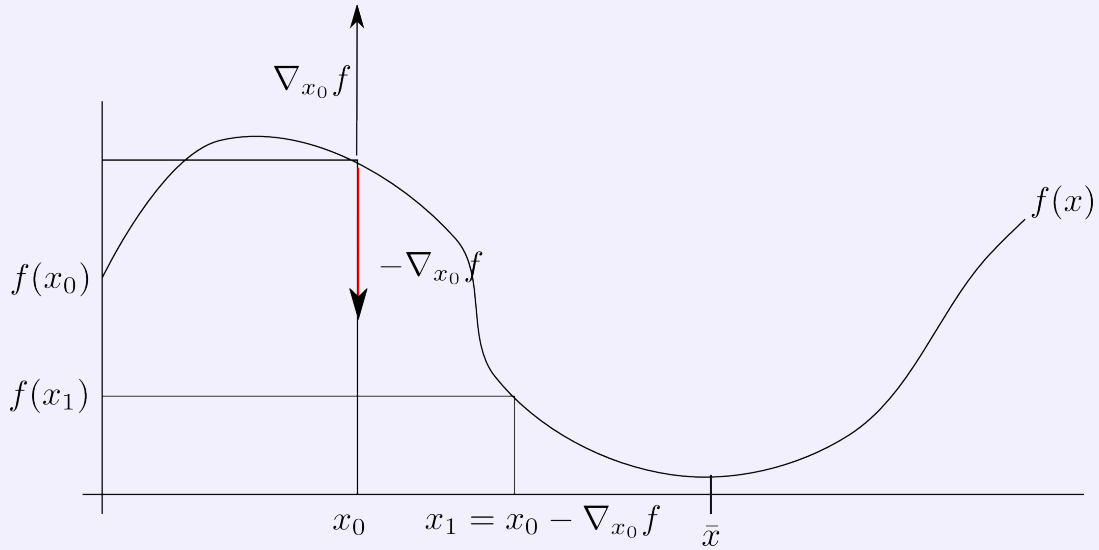


Segmentation of heart using a variant of Snakes by A. Tatu, F. Lauze, S. Sommer and N. Nielsen. External contour is the initial one. Internal after convergence.

Outline

Gradient Descent

- $\bar{x}$ minimizes $f(x)$, f differentiable: $\nabla_{\bar{x}} f = 0$
- Gradient indicates the direction to take so as make the function increase most. For the opposite (i.e., getting to a minimum): follow $-\nabla f$



- Descent: Take small, steps in the direction indicated by $-\nabla f$: embed an original estimate x_0 in a family of points x_i (or $x(t)$ for a continuous formulation)

$$x_{i+1} = x_i - \tau \nabla_{x_i} F, \quad \frac{dx}{dt} = -\nabla_{x(t)} f$$

Simple Example – Explicit Scheme

- $F(x) = x^2$. $\nabla_x F = F'(x) = 2x$. Minimum for $x = 0$.
- Gradient descent:

$$x_{i+1} = x_i - \tau F'(x_i) = x_i - 2\tau x_i = (1 - 2\tau)x_i$$

- If τ too large, problems: try with $\tau = 1$!
- If $\tau < 1/2$, decreasing sequence.
- If $\tau = 1/2$, converges in 1 step!

Nonlinear Problem

- $F(x) = F(x_1, \ldots, x_n) = \|x\|$, $x \in \mathbb{R}^n$, $\|x\| = \sqrt{x_1^2 + \ldots x_n^2}$.
- Minimum at $x = (0, \ldots, 0)$.
- Gradient of F:

$$\nabla_x F = \begin{pmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{x_1}{\|x\|} \\ \vdots \\ \frac{x_n}{\|x\|} \end{pmatrix} = \frac{x}{\|x\|}$$

- Explicit descent:

$$x_{i+1} = x_i - \tau \frac{x_i}{\|x_i\|} = x_i \left(1 - \frac{\tau}{\|x\|}\right)$$

- If $\tau > \|x_i\|$, $1 - \frac{\tau}{\|x\|} < 0$, oscillating!

Semi-implicit scheme

- $\nabla_x F = \frac{x}{\|x\|}$ is complex, several ways to deal with it in descent.
- Instead of explicit scheme, semi-implicit:

$$x_{i+1} = x_i - \tau \frac{x_{i+1}}{\|x_i\|}$$

- Provides descent scheme:

$$x_{i+1} = \frac{x_i}{1 + \tau \|x_i\|}$$

- Will never oscillate!
- If possible, use a semi-implicit scheme instead of an explicit one.

Solving the segmentation

- This is done iteratively by moving the current estimate $C^s = (P_1^s = (x_1^2, y_1^s), P_2^s = (x_2^2, y_2^s), \ldots, P_n^n = (x_n^s, y_n^s))$ of the contour to the next one $C^{s+1} = (P_1^{s+1}, P_2^{s+1}, \ldots, P_n^{s+1})$ in such a way that the snake energy decreases most. This needs small steps.
- The theory shows that a solution can be described as solution of the systems of linear equations

$$M \begin{pmatrix} x_1^{s+1} \\ x_2^{s+1} \\ \vdots \\ x_N^{n+1} \end{pmatrix} = \begin{pmatrix} x_1^s - \gamma F_x(x_1^s, y_1^s) \\ x_2^s - \gamma F_x(x_2^s, y_2^s) \\ \vdots \\ x_n^s - \gamma F_x(x_n^s, y_n^s) \end{pmatrix} \qquad M \begin{pmatrix} y_1^{s+1} \\ y_2^{s+1} \\ \vdots \\ y_N^{s+1} \end{pmatrix} = \begin{pmatrix} y_1^s - \gamma F_y(x_1^s, y_1^s) \\ y_2^s - \gamma F_y(x_2^s, y_2^s) \\ \vdots \\ y_n^s - \gamma F_y(x_n^s, y_n^s) \end{pmatrix}$$

- F_x and F_y are the $x-$ and $y-$derivatives of the external force F, described later
- M is the system matrix (coefficients of the systems of equations) described in an example in the slide next to the following one.

- Solving the system of equations of the previous slide means to compute the new values x_i^{s+1} and y_i^{s+1} as follows:

$$\begin{pmatrix} x_1^{s+1} \\ x_2^{s+1} \\ \vdots \\ x_N^{n+1} \end{pmatrix} = M^{-1} \begin{pmatrix} x_1^s - \gamma F_x(x_1^s, y_1^s) \\ x_2^s - \gamma F_x(x_2^s, y_2^s) \\ \vdots \\ x_n^s - \gamma F_x(x_n^s, y_n^s) \end{pmatrix}, \quad \begin{pmatrix} y_1^{s+1} \\ y_2^{s+1} \\ \vdots \\ y_N^{s+1} \end{pmatrix} = M^{-1} \begin{pmatrix} y_1^s - \gamma F_y(x_1^s, y_1^s) \\ y_2^s - \gamma F_y(x_2^s, y_2^s) \\ \vdots \\ y_n^s - \gamma F_y(x_n^s, y_n^s) \end{pmatrix}$$

- M^{-1} is the inverse matrix of M.
- This is the same matrix for both the abscissas and ordinates.

System Matrix

- Set $A = \tau\beta$, $B = -\tau(\alpha + 4\beta)$, $C = 1 + \tau(2\alpha + 6\beta)$,
- Here is the matrix of the linear system for a curve represented by 10 points.

$$M = \begin{pmatrix} C & B & A & 0 & 0 & 0 & 0 & 0 & A & B \\ B & C & B & A & 0 & 0 & 0 & 0 & 0 & A \\ A & B & C & B & A & 0 & 0 & 0 & 0 & 0 \\ 0 & A & B & C & B & A & 0 & 0 & 0 & 0 \\ 0 & 0 & A & B & C & B & A & 0 & 0 & 0 \\ 0 & 0 & 0 & A & B & C & B & A & 0 & 0 \\ 0 & 0 & 0 & 0 & A & B & C & B & A & 0 \\ 0 & 0 & 0 & 0 & 0 & A & B & C & B & A \\ A & 0 & 0 & 0 & 0 & 0 & A & B & C & B \\ B & A & 0 & 0 & 0 & 0 & 0 & A & B & C \end{pmatrix}$$

- Note the values at the upper-right and lower down corners. This is because we deal with closed curves.

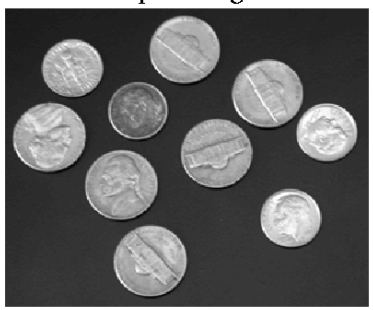
System Matrix II

- Constructing the system matrix is easy. Python possesses a `roll()` function in `numpy` that rolls and wraps the content of a vector.
- Each line can be obtained from the previous by proper rolling! Easy!
- Matlab has a command called `toeplitz()` that can construct the system matrix very easily.
- When the system matrix has been computed, it can be immediately inverted. `numpy.linagl.inv()` does the job for python, while `inv()` will do it for Matlab.

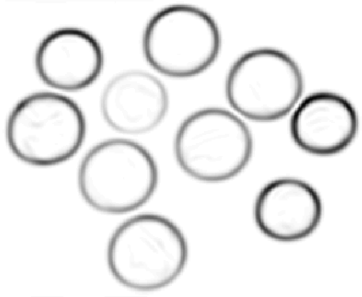
Derivatives of external forces

An example using the one described before.

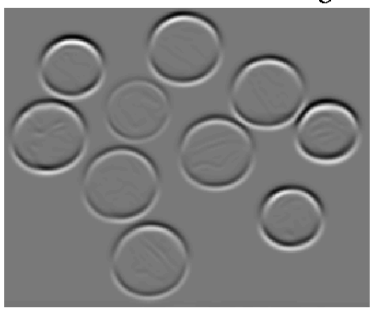
Input image



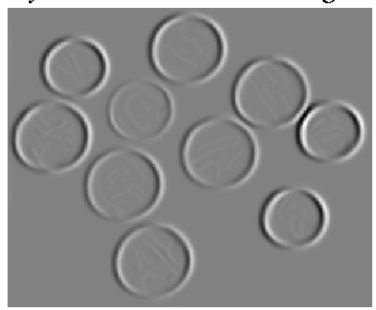
Force image



x-derivative of force image



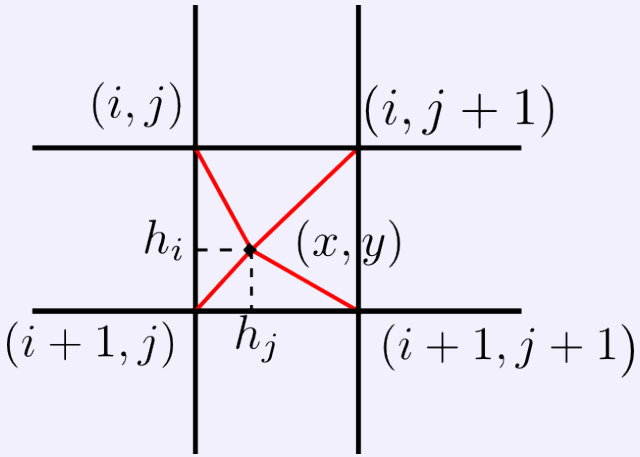
y-derivative of force image



- A direct calculation shows that for $F = -\|\nabla I_\sigma\|^2$,

$$F_x = -2\left(I_{\sigma x} I_{\sigma xx} + I_{\sigma y} I_{\sigma xy}\right), \quad F_y = -2\left(I_{\sigma x} I_{\sigma xy} + I_{\sigma y} I_{\sigma yy}\right).$$

- Main difficulty in computing F_x (or F_y) at (x, y): values of F_x generally only known at grid position (i, j), $i = 0 \ldots M-1$, $j = 0 \ldots N-1$ with $M \times N$ size of image I (and also of F, F_x and F_y). But (x, y) might not be on the grid. Need for interpolation.



$$\begin{aligned} F(x,y) \approx\ & F(i,j)(1-h_i)(1-h_j) \\ & + F(i+1,j)h_i(1-h_j) \\ & + F(i,j+1)(1-h_i)h_j \\ & + F(i+1.j+1)h_i h_j \end{aligned}$$

Bilinear Interpolation

- A good choice: bilinear or bicubic interpolation. Matlab built-in function `interp2` does it. I have provided a Python class to do it.

Outline

External forces

- The original paper on Snakes mentions many possible external forces. We will only deal with 2 of them.
- $F(I) = -|g_\sigma \nabla I|^2$. This corresponds to the one used by the Canny edge detector.
- The Marr-Hildreth edge detector detect edges at the zero-crossings of the Laplacian of Gaussian $g_\sigma \Delta I$. To make an external force out of it, we take

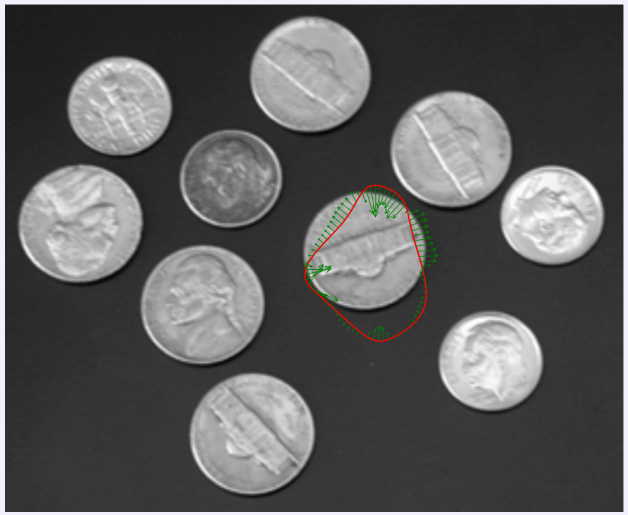
$$F(I) = -\left(g_\sigma \Delta I\right)^2 .$$

This will become small at the zeros-crossings of the Laplacian, and large outside.

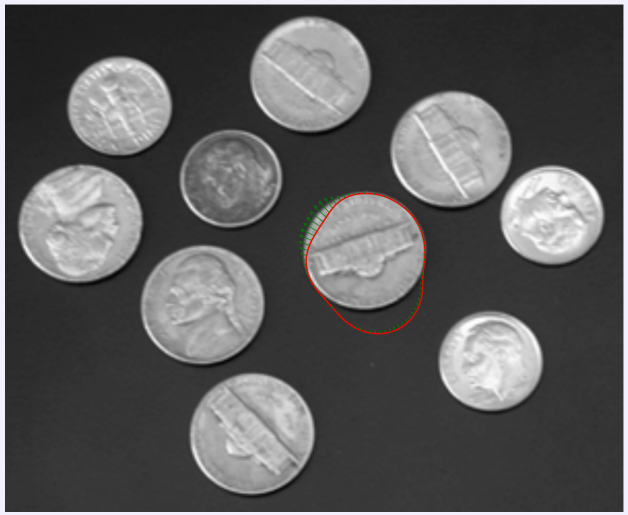
External forces II

- In order to implement the snakes, on must compute the partial derivatives of F, $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$.
- They can be computed as functions of (higher-order) derivatives of the image I (or $g_\sigma \star I$).
- These higher order derivatives can be computed directly numerically with the finite differences mentioned in the first slides, or using proper derivatives of Gaussians. Note that Python as a way to do it via the `scipy.ndimage.filters.gaussian_filter`.
- The resulting $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$ are images, defined on a grid of pixels. But there is no guaranty that points of the curve are on the grid. This means that to evaluate a quantity such as $\frac{\partial F}{\partial x}(x_i, y_i)$ interpolation will be necessary.

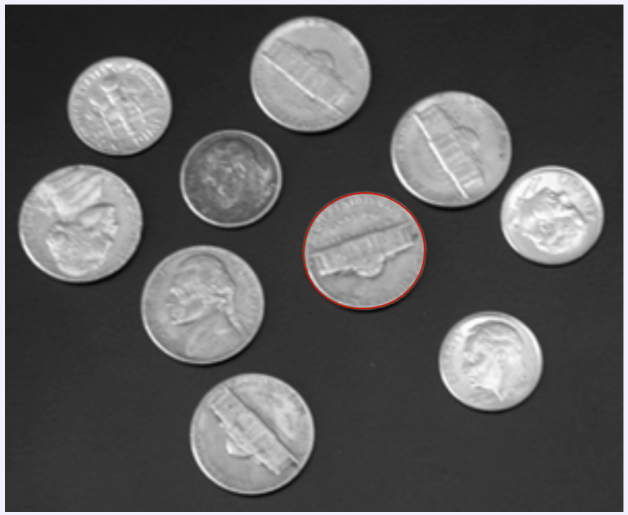
Snake with Balloon Forces: Example



Initial curve and forces.



During evolution.



At convergence.

About the assignment

- The assignment consists in implementing the implicit scheme for snakes and experimenting with it (one can start by an explicit one, but the added complexity of the explicit scheme is small).
- This should have a certain level of interactivity so it can easily be tested. Python and Matlab have proper support for that.