Active Contours

Thomas Breuel

UniKL

Active Contours

The idea behind snakes or active contour segmentation is the following:

- start with a contour that represents an approximation of the segmentation
- deform the contour slightly in order to improve the segmentation

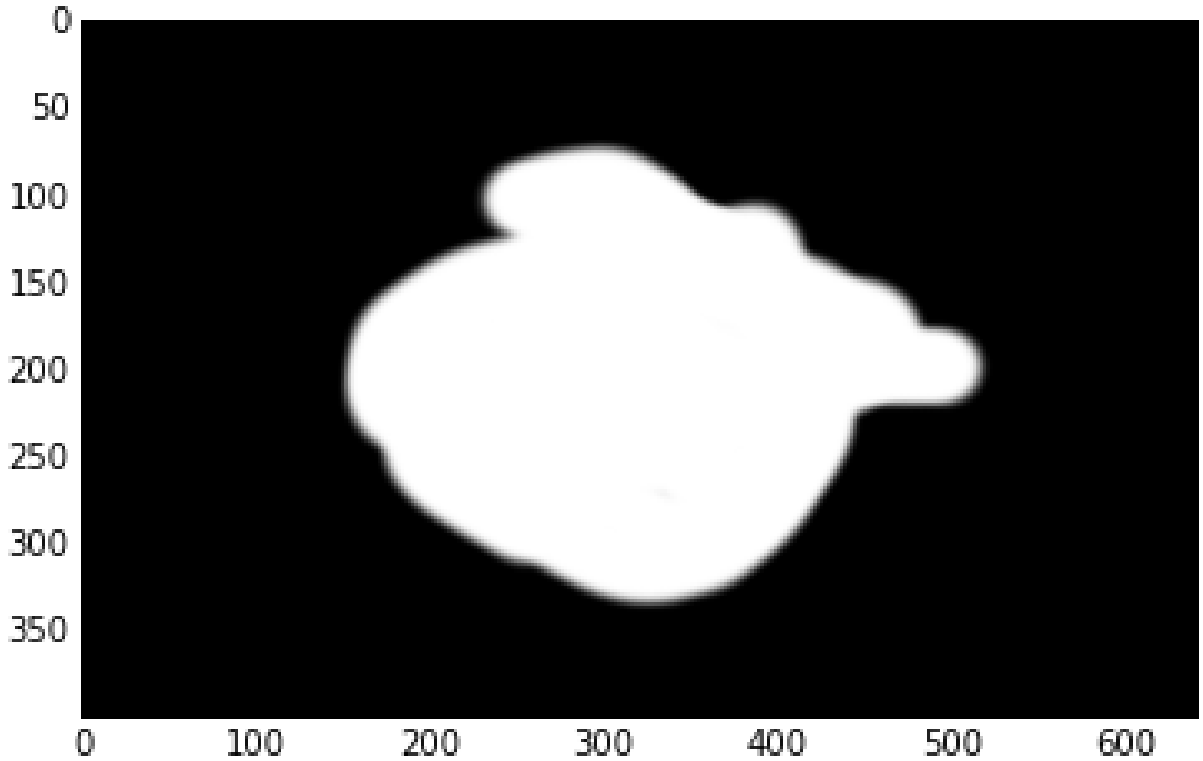
Deformation of the contour may be done by a number of criteria:

- make the contour conform closer to edges in the image
- make the inside and outside of the regions more homogeneous

GVF Snakes

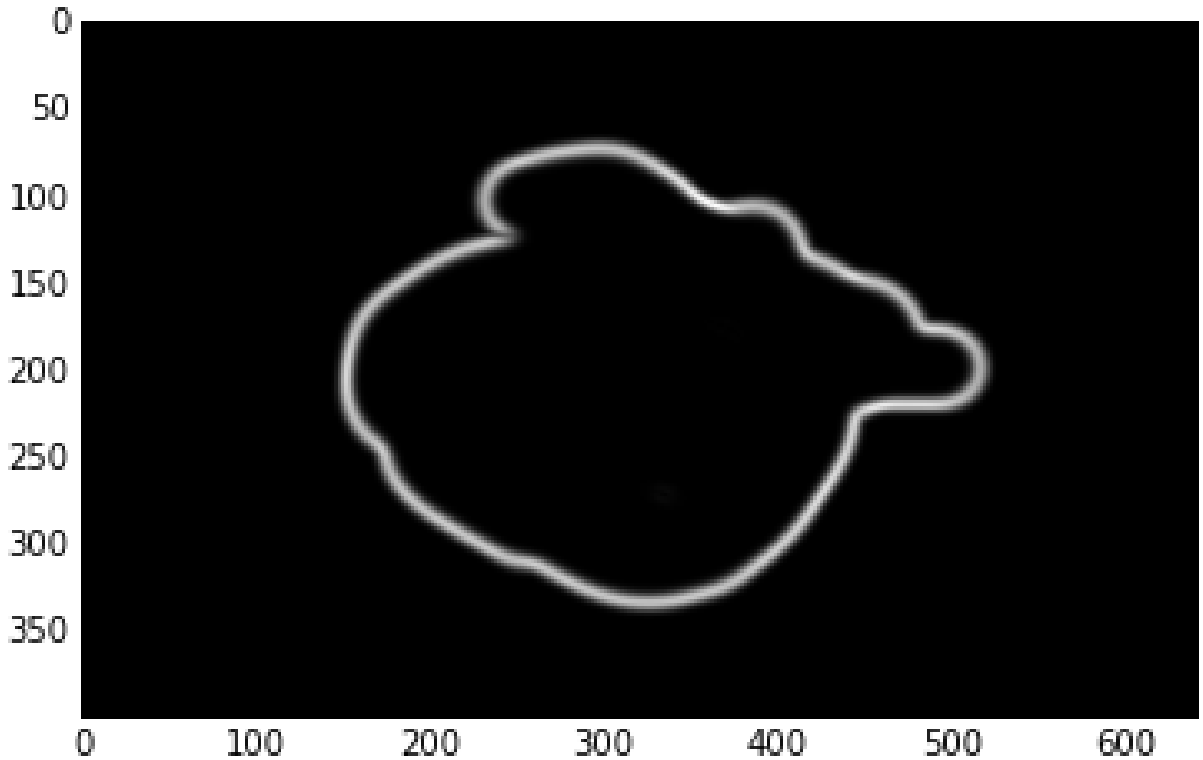
Let's illustrate another image segmentation method with a really simple example (of course, we could solve this one just by thresholding).

```
1 blob = mean(imread("blob.png"),2)
2 imshow(blob)
```



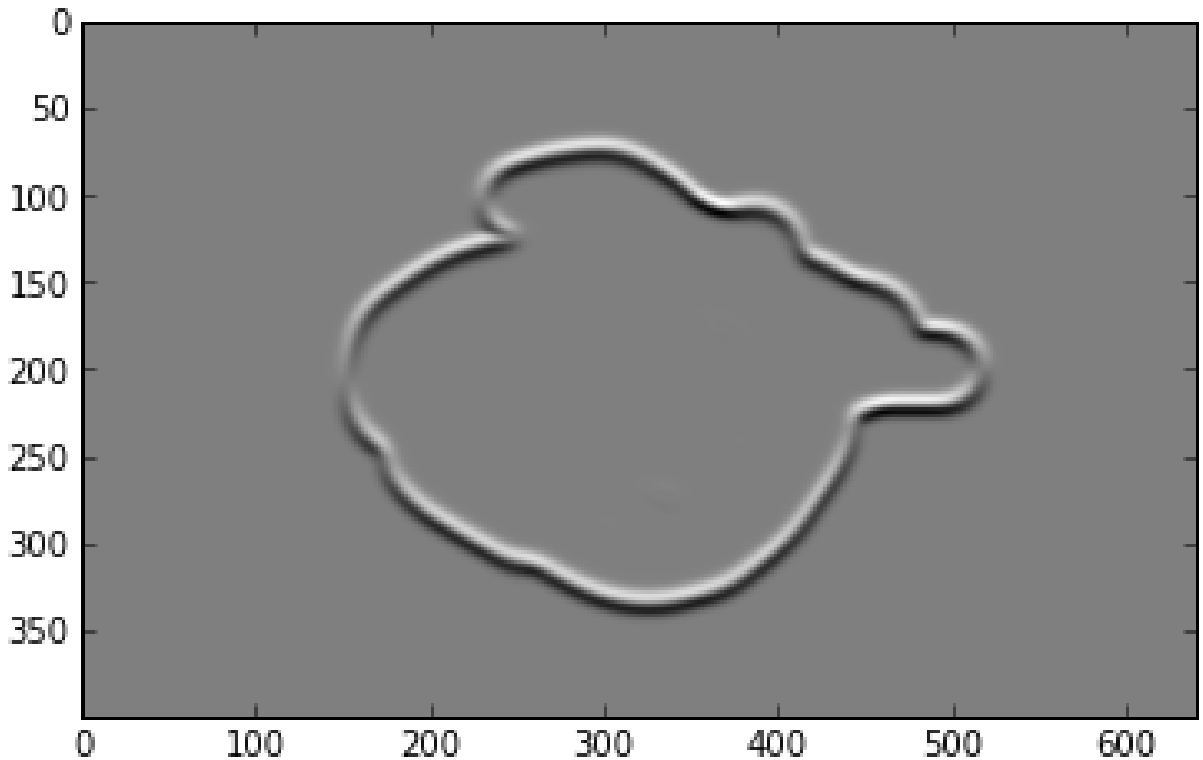
We actually perform the segmentation on the edge image. Snake based segmentation works even if the edges do not form a closed contour.

```
f = filters.gaussian_gradient_magnitude(blob,2.0)
imshow(f)
```



Now we compute the gradients of the edges. These gradients point towards the center of the edge.

```
fx = filters.gaussian_filter(f,2.0,(1,0))
fy = filters.gaussian_filter(f,2.0,(0,1))
imshow(fx)
```



Gradient Vector Flow

getting long range flow information

These gradients are very short-range. But we will want to use them to draw a curve towards the center of the edges.

In order to do this, we need to propagate or spread out the gradients.

The GVF approach accomplishes this by using generalized diffusion.

interpolating missing gradient information

The starting point is the derivatives $g = (f_x, f_y)$.

We want a spread out vector field $\phi = (u, v)$ that agrees with g where g is large and interpolates elsewhere.

$$E = \int \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) \; + \; |g|^2 \; |\phi - g|^2 dxdy$$

Here, the first term is a smoothness term.

The second term is the difference between ϕ and g scaled by the magnitude of g .

iterative solution by diffusion

This can be solved using a diffusion equation (that can be formally shown using the calculus of variations).

This gives rise to the following Euler equations:

$$\mu\nabla^2 u - (u - f_x)|g|^2 = 0$$
$$\mu\nabla^2 v - (v - f_y)|g|^2 = 0$$

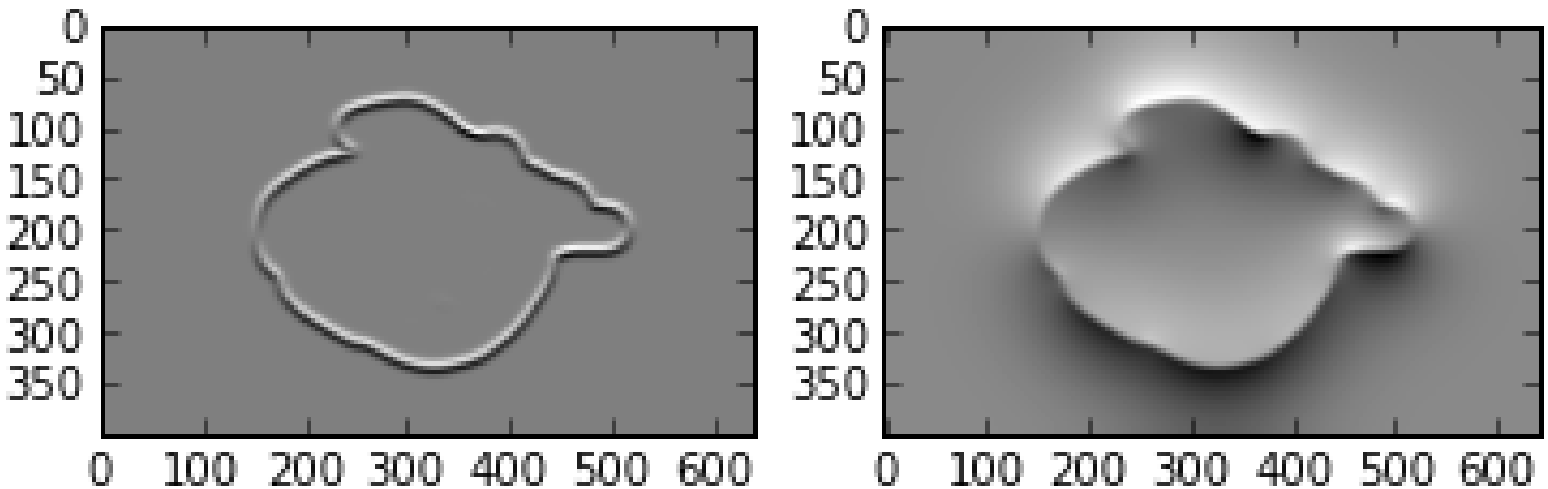
These equations can be solved iteratively by diffusion:

$$u_t = \mu\nabla^2 u - (u - f_x)|g|^2$$
$$v_t = \mu\nabla^2 v - (v - f_y)|g|^2$$

```
u = fx.copy()
v = fy.copy()
mu = 0.1
for i in range(10000):
    m = (fx**2+fy**2)
    ut = mu*filters.laplace(u)-(u-fx)*m
    vt = mu*filters.laplace(v)-(v-fy)*m
    u += ut
    v += vt
```

result of GVF gradient spreading

```
1 subplot(121); imshow(fx)
2 subplot(122); imshow(u)
```



Snake Fitting

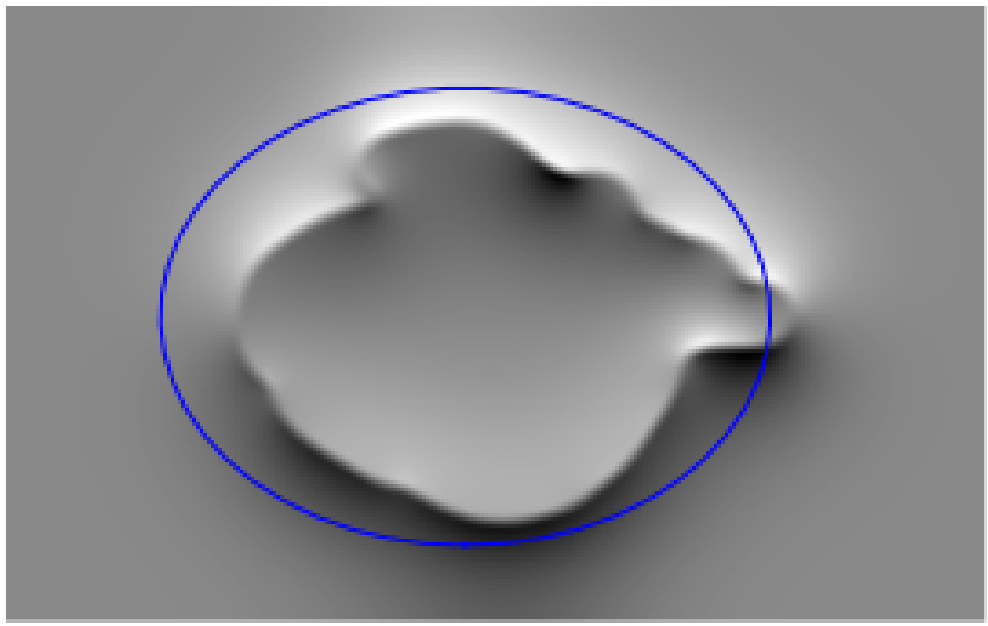
Let's generate a closed contour and plot it.

We consider the closed contour a function $p : [0, 1] \rightarrow R^2$

For actual computations, we subsample this path.

```
t = linspace(0,2*pi,50)
path = array([150*cos(t)+200,200*sin(t)+300]).T
imshow(u); plot(path[:,1],path[:,0]); axis("off")
```

```
(-100.0, 700.0, 400.0, -50.0)
```



The simplest kind of fitting is to update the curve iteratively by:

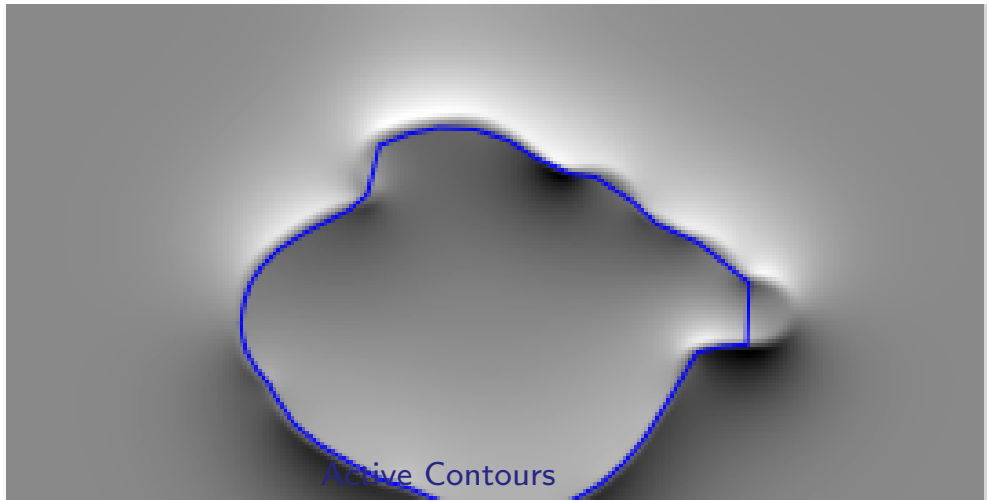
$$p_t = \phi$$

This means every point of the curve is just drawn in by the vector field, and eventually comes to lie directly on the edge.

iterative deformation of contour by GVF

```
t = linspace(0,2*pi,50)
path = array([150*cos(t)+200,200*sin(t)+300]).T
for t in range(10000):
    dx = array([u[int(y),int(x)] for y,x in path])
    dy = array([v[int(y),int(x)] for y,x in path])
    delta = array([dx,dy]).T
    path += 30.0*delta
imshow(u); plot(path[:,1],path[:,0]); axis("off")
```

```
(-100.0, 700.0, 400.0, -50.0)
```



smoothness constraints

Usually, we want to impose smoothness constraints. This is actually again a kind of diffusion equation.

For example, we can update as:

$$p_t = \phi + \alpha p''$$

Here, p'' is the second derivative with respect to the path length.

This update rule will have the effect of diffusing the positions of points", i.e., smoothing the curve. We can do the same for second derivatives, i.e., curvatures, and end up with:

$$p_t = \phi + \alpha p'' - \beta p''''$$

deformation with smoothness term

```
t = linspace(0,2*pi,50)
path = array([150*cos(t)+200,200*sin(t)+300]).T
for t in range(30000):
    x2 = filters.gaussian_filter(path,1.0,(2,0),mode='wrap')
    x4 = filters.gaussian_filter(x2,1.0,(2,0),mode='wrap')
    dx = array([u[int(y),int(x)] for y,x in path])
    dy = array([v[int(y),int(x)] for y,x in path])
    delta = array([dx,dy]).T
    path += 0.001*x2-0.001*x4+10.0*delta
imshow(u); plot(path[:,1],path[:,0]); axis("off")
```

```
(-100.0, 700.0, 400.0, -50.0)
```