Lecture 21: Image Understanding

Last modified on March 20, 2000 at 6:00 PM

Contents

Reading

SH&B, Chapter 8 (primarily 8.1–8.3)

The material in Section 8.4 is covered in more detail in CS 521.

21.1 Introduction

So far, we've looked at low-level processing (clean-up, edge or other feature detection, etc.), segmenting the image into regions that hopefully correspond to objects, and representing those objects using various representations. *Image understanding* is the process of actually interpreting those regions/objects to figure out what's actually happening in the image. This may include figuring out what the objects are, their spatial relationship to each other, etc. It may also include ultimately making some decision for further action.

21.2 Control Strategies

21.2.1 Bottom-Up

The process as we've just described it is *bottom-up*: it starts from the raw image data, works upward to object shape representation, and from there to a higher-level analysis or decision.

21.2.2 Top-Down

Image understanding can also work from the top down. Such processing makes hypotheses about that is happening, then uses the image data to validate/reject those hypotheses. Most of these approaches are *model-based*. That is, they have an approximate model of what they think they're looking at, then try to fit that model to the data. In a sense, primitive-fitting approaches such as the Hough transform used this idea. This idea can be extended further to so-called *deformable models*, in which you can deform the model to better fit the data. The goodness of the match is the inverse of how much you have to work to deform the model to fit the data.

21.2.3 Hybrid Hierarchical Control

Obviously, these two paradigms aren't mutually exclusive: one might use bottom-up processing to bring the information to something higher than the pixel level, then switch to top-down processing using this intermediate representation to validate or reject hypotheses.

21.2.4 Active Vision

A third alternative, neither quite bottom-up or top-down, presents itself when the vision system is part of a larger system capable of acting so as to be able to influence the position/view/etc. Such *active vision* approaches more accurately model the way people interact with the world. The idea is to make a tentative hypothesis (using either top-down or bottom-up processing) then ask yourself, "based on what I already know (and suspect), what do I need to do to be able to acquire the information that will best help me analyze the image or otherwise accomplish my task?"

21.3 Active Contours (Snakes)

The earliest and best known active contour approach is *snakes*: deformable splines that are acted upon by image, internal, and user-defined "forces" and deform to minimize the "energy" they exert in resisting these forces.

Your text has a good description of snakes, and you should also read the original paper (Kass, Witken, and Terzopolous in *CVPR'87*).

Notice that the general form of a snake follows the idea introduced earlier when we discussed graph-based approaches:

1. Establish the problem as the minimization of some cost function.
2. Use established optimization techniques to find the optimal (minimum cost) solution.

In the case of a snake, the cost function is the "energy" exerted by the snake in resisting the forces put upon it. The original formulation of this "energy" was

$$E_{\text{snake}} = w_{\text{int}}E_{\text{int}} + w_{\text{image}}E_{\text{image}} + w_{\text{con}}E_{\text{con}} \tag{21.1}$$

where each term is as follows:

E_{int}	Internal Energy	Keeps the snake from bending too much
E_{image}	Image Energy	Guides the snake along important image features
E_{con}	Constraint Energy	Pushes or pulls the snake away from or towards user-defined positions

The total energy for the snake is the integral of the energy at each point:

$$E^*_{\text{snake}} = \int_0^1 E_{\text{snake}}(s)\ ds \tag{21.2}$$

21.3.1 Internal Energy

The *internal energy* term tries to keep the snake smooth. Such smoothness constraints are also a common theme in computer vision, occurring in such approaches as

- Bayesian reconstruction
- Shape from shading
- Stereo correspondence
- and many others ...

The internal energy term in general keeps the model relatively close to its original *a priori* shape. In this case, we explicitly assume that the contour we want is generally smooth but otherwise unconstrained. Other models might start with an approximation of a brain, a heart, a kidney, a lung, a house, a chair, a person, etc.orm this model—in all cases, the internal energy term constrains the deformation.

One has to be careful with the weighting given to internal energy terms, though: too much weight means the model stays too “rigid” and the system “sees what it wants to see”, too little weight means the model is too flexible and can pretty much match up to anything.

In the original snakes implementation, they used two terms to define the internal energy: one to keep the snake from stretching or contracting along its length (elasticity) and another to keep the snake from bending (curvature):

$$E_{\text{int}}(s) = \alpha(s)\left\|\tfrac{d\bar{v}}{ds}(s)\right\|^2 + \beta(s)\left\|\tfrac{d^2\bar{v}}{ds^2}(s)\right\|^2$$

Notice that both $\alpha(s)$ and $\beta(s)$ are functions of the arc length along the snake. This means that we can (perhaps interactively), keep the snake more rigid in some segments and more flexible in others.

21.3.2 Image Energy

The *image energy* term is what drives the model towards matching the image. It is usually inversely based on image intensity (bright curve finding), gradient magnitude (edge finding), or similar image features. Make sure to note the inverse relationship: strong features are *low* energy, and weak features (or no features) are *high* energy.

An interesting image energy term used in the original snakes paper also tracked *line terminations*. These are useful in analyzing visual illusions such as the Kinisa and Ehringhaus illusions illustrated in your text. This line termination term is the same level-set curvature measure that we learned about when we studied differential geometry (and which you expanded in HW 5).

21.3.3 Constraint Energy

Some systems, including the original snakes implementation, allowed for user interaction to guide the snakes, not only in initial placement but also in their energy terms. Such *constraint energy* can be used to interactively guide the snakes towards or away from particular features.

The original snakes paper used “springs” (attraction to specified points) and “volcanos” (repulsion from specified points).

21.3.4 Implementation

The simplest form of optimization is *gradient-descent minimization*. The idea is to find the minimum of a function f by iteratively taking a step “downhill”.

For example, let’s consider a function of only one variable. If we have a starting guess at the value of the solution, we can look at the slope at that point and decide to increment our solution (negative slope) or decrement our solution (positive slope). Notice the negation there: if the slope is positive, downhill is *backwards*; and if the slope is negative, downhill is *forwards*. We can thus implement gradient-descent minimization as

$$x_{t+1} = x_t - \gamma \frac{df}{dx}(x_t)$$

where γ controls the size of the step at each iteration.

For functions of two dimensions, the fastest direction downhill is the opposite of the fastest direction uphill (the gradient). This is basically just the same as doing gradient-descent minimization in each variable at the same time:

$$x_{t+1} = x_t - \gamma \frac{df}{dx}(x_t)$$

and

$$y_{t+1} = y_t - \gamma \frac{df}{dy}(y_t)$$

Or, more generally for any function of a vector $\overline{x}$:

$$\overline{x}_{t+1} = \overline{x}_t - \gamma \nabla f(\overline{x}_t)$$

Implementing such minimization in this form is actually quite simple:

```
grad = CalculateGradient(f,x);
while (magnitude(grad) > convergence_threshold) {
    x -= gamma * grad;
    grad = CalculateGradient(f,x);
}
```

The difficult part isn't implementing the minimization, it's differentiating the function you're trying to minimize.

In the case of Eq. 21.2, we can approximate it using a number of discrete points on the snake $\bar{v}_i = (x_i, y_i)$:

$$E^*_{\text{snake}} \approx \sum_1^n E_{\text{snake}}(\bar{v}_i)$$

what's nice about this is that the derivative of a sum is the sum of the derivatives, so

$$\begin{aligned}\nabla E^*_{\text{snake}} &\approx \nabla \left[\sum_1^n E_{\text{snake}}(\bar{v}_i)\right] \\ &= \sum_1^n \nabla E_{\text{snake}}(\bar{v}_i)\end{aligned}$$

We can thus think of the problem as iteratively adjusting each of the points $\bar{v}_i$ using its own gradient-descent minimization:

$$\bar{v}_i \leftarrow \bar{v}_i - \nabla E_{\text{snake}}(\bar{v}_i)$$

Using Eq. 21.1, we get

$$\begin{aligned}\nabla E_{\text{snake}}(\bar{v}_i) &= \nabla \left[w_{\text{int}} E_{\text{int}}(\bar{v}_i) + w_{\text{image}} E_{\text{image}}(\bar{v}_i) + w_{\text{con}} E_{\text{con}}(\bar{v}_i)\right] \\ &= w_{\text{int}} \nabla E_{\text{int}}(\bar{v}_i) + w_{\text{image}} \nabla E_{\text{image}}(\bar{v}_i) + w_{\text{con}} \nabla E_{\text{con}}(\bar{v}_i)\end{aligned}$$

Notice that $w_{\text{image}} \nabla E_{\text{image}}(\bar{v}_i) + w_{\text{con}} \nabla E_{\text{con}}(\bar{v}_i)$ depends only on the image, not on the relationship of the snake to any other part of itself, so we can precalculate this for every point in the image. Simply calculate $w_{\text{image}} E_{\text{image}} + w_{\text{con}} E_{\text{con}}$ everywhere, then measure its derivatives locally. Let's call this ∇E_{ext}:

$$\nabla E_{\text{ext}} = w_{\text{image}} \nabla E_{\text{image}} + w_{\text{con}} \nabla E_{\text{con}}$$

Substituting, our minimization now becomes

$$\bar{v}_i \leftarrow \bar{v}_i - \gamma \left[w_{\text{int}} \nabla E_{\text{int}}(\bar{v}_i) + \nabla E_{\text{ext}}(\bar{v}_i)\right]$$

So, the only thing left to do is to solve for the gradient of the internal energy. Unfortunately, this is rather complicated since it's a function of the spline itself, not the image. Fortunately, it simplifies considerably if $\alpha(s)$ and $\beta(s)$ are constant—Kass, Witkin, and Terzopolous published the following:

$$\begin{aligned}\nabla E_{\text{int}}(s) &= \nabla \left[\alpha \left\| \frac{d\bar{v}}{ds}(s) \right\|^2 + \beta \left\| \frac{d^2\bar{v}}{ds^2}(s) \right\|^2\right] \\ &= \left[\alpha \nabla \left\| \frac{d\bar{v}}{ds}(s) \right\|^2 + \beta \nabla \left\| \frac{d^2\bar{v}}{ds^2}(s) \right\|^2\right] \\ &= \alpha \frac{\partial^2 \bar{v}}{\partial s^2} + \beta \frac{\partial^4 \bar{v}}{\partial s^4}\end{aligned}$$

These can be approximated using finite differences—the second derivative w.r.t. s can be calculated using three adjacent points on the snake, and the fourth derivative w.r.t. s can be calculated using five adjacent points. It also helps to separate the x and y components.

Putting it all together:

$$\bar{v}_i \leftarrow \bar{v}_i - \gamma \left\{ w_{\text{int}} \left[\alpha \frac{\partial^2 \bar{v}}{\partial s^2}(\bar{v}_i) + \beta \frac{\partial^4 \bar{v}}{\partial s^4}(\bar{v}_i) \right] + \nabla E_{\text{ext}}(\bar{v}_i) \right\}$$

or more simply:

$$x_i \leftarrow x_i - \gamma \left\{ w_{\text{int}} \left[\alpha \frac{\partial^2 x}{\partial s^2}(\bar{v}_i) + \beta \frac{\partial^4 x}{\partial s^4}(\bar{v}_i) \right] + \frac{\partial}{\partial x} E_{\text{ext}}(\bar{v}_i) \right\}$$

and

$$y_i \leftarrow y_i - \gamma \left\{ w_{\text{int}} \left[\alpha \frac{\partial^2 y}{\partial s^2}(\bar{v}_i) + \beta \frac{\partial^4 y}{\partial s^4}(\bar{v}_i) \right] + \frac{\partial}{\partial y} E_{\text{ext}}(\bar{v}_i) \right\}$$

Got all that? Before you start the iteration, precalculate $E_{\text{ext}}(\bar{v}_i)$ and calculate the derivatives of this w.r.t. x and y separately. When you're ready to start the iteration, calculate at each point $\frac{\partial^2 x}{\partial s^2}(\bar{v}_i)$ and $\frac{\partial^2 y}{\partial s^2}(\bar{v}_i)$ using three adjacent points and $\frac{\partial^4 x}{\partial s^4}(\bar{v}_i)$ and $\frac{\partial^4 y}{\partial s^4}(\bar{v}_i)$ using five adjacent points. Then calculate the incremental change in the x and y components of each point v_i. You then have to *recalculate* the derivatives of the internal energy on each iteration. For the external energy term, you can simply read from the precalculated images for the derivatives of $E_{\text{ext}}(\bar{v}_i)$ using the new positions of each v_i.

21.4 Point-Density Models

Point density models can be thought of as deformable models in which the deformation "energy" is based on statistical properties of a large number of training examples. This has the powerful advantage of *allowing deformation where the objects themselves normally differ while remaining more rigid where the objects themselves are normally consistent*.

First identify a number of key *landmark* points for the object. These need not be on the contour but can be *any* identifiable points.

Now, gather a collection of sample images with varying shapes that you want to recognize. (For example, if you're building a system to recognize and analyze brains, get a collection of sample brains; if you're building a system to recognize different kinds of fish, get yourself samples of each kind of fish; etc.) For each image in your training set, find the landmarks and store their locations. Now, you need to register these images by transforming (translating, rotating) the landmark points so that they are all registered relative to a common *mean shape*.

We can now measure the covariance matrix for all of the landmarks across all of the training shapes. This tells us not only the consistency of each landmark but *the way each landmark tends to move as the others move*.

The covariance matrix can be further analyzed by computing its eigenvalues and eigenvectors (called *principal component analysis*). These eigenvectors are called the *modes of variation* of the shapes, and the eigenvalues tell you the rigidity or flexibility along these modes. Notice that a mode is *not* a single direction—it's an overall change in all of the landmarks relative to each other.

Point-density or landmark-based models, though they can be computationally expensive, are among the most powerful models for shape description and deformation currently in use.

Vocabulary

- Bottom-up processing
- Top-down processing
- Active vision
- Active contours / Snakes
- Gradient-descent minimization

- Point-density models
- Landmarks