Image Formation and Analysis
(Formation et Analyse d'Images)

James L. Crowley

ENSIMAG 3 - MMIS Option MIRV First Semester 2010/2011

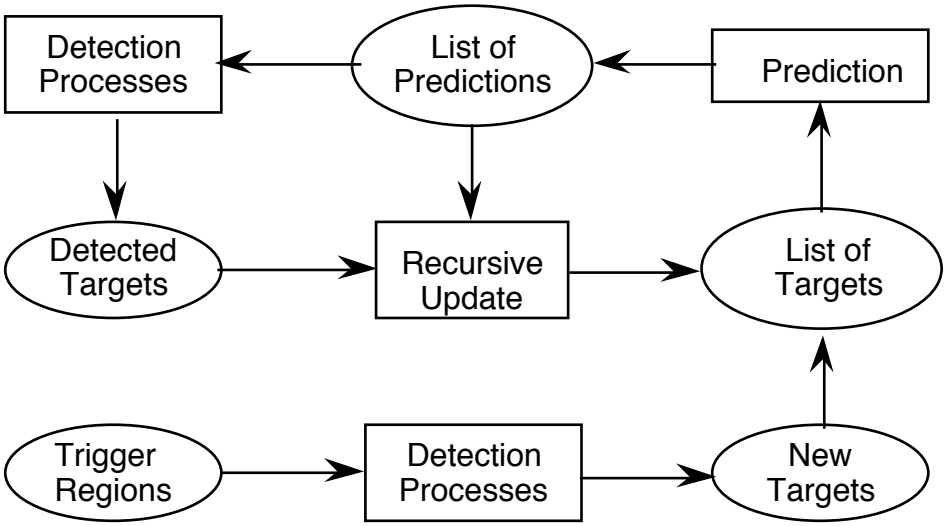
Lesson 10 17 Jan 2011

Bayesian Tracking of Blobs

Outline :

Bayesian Detection and Tracking

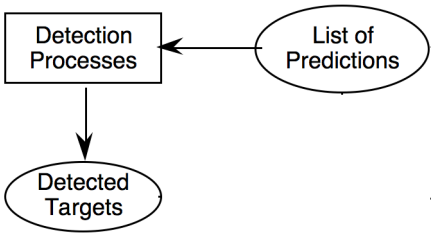
Architecture of a Baysian Tracking Process



A Bayesian Tracker is a cyclic process composed of the cycles predict, detect and update.

Tracking
1) Focusing computing resources
2) Improves reliability by compensating for lost detections
3) Improves precision by avoiding distractors
4) Makes it possible to estimate motion derivatives.

We will use a Bayesian tracker with pixel level detection.
Let us start with pixel level target detection:



Pixel Level Target Detection

A pixel level detector estimates the likelihood that a pixel belongs to a target. Many possible estimation processes can be used. Popular pixel level detectors include:

1) Ratio of Color Histograms
2) Background Difference Subtraction
3) Motion (temporal image differences).
4) Ratio of Receptive field histograms

We have recently combined a Bayesian tracker with face detection using a Cascade of Linear Detectors. In this case, we used the cascade level at which detection failed as a likelihood estimator for each pixel.

Background Difference Subtraction

Background difference subtraction estimated the likelihood that a pixel is target by subtracing the current image p(i,j) from an estimated empty background image, $b_t(i,j)$.

$$d(i,j) = p(i,j) - b_t(i,j)$$

There are many techniques for estimating the empty backgound image.

If the camera is stationary, and the illumination does not change, then the background image can be taken by observing an empty scene.

In most scenes the illumination changes.
If the camera is stationary, but the illumination changes, it is possible to estimate the empty background using a recursive estimation:

To avoid corrupting the background with targets, we use a mask to detect regions where targets are predicted, f(i,j). The mask is set to zero (false) wherever a target is predicted or detected and true everywhere else.

$$\forall_{i,j} \text{ if } f(i,j) \text{ then } b_t(i,j) = a \cdot b_{t-1}(i,j) + (1-a) \cdot p(i,j)$$

A typical value of a would be 0.999. The value can be adjusted to reacted fasters to slower.

A problem with this method is motion by non-target object, such as furniture.

Classification by Ratio of Histograms of pixel values

Histograms provide an alternate view of Bayes Rule.

Histograms

As we saw, for integer x from a bounded set of values, such that $x \in [x_{min}, x_{max}]$,

the probability that a random observation X takes on x is

$$P(X=x) = \frac{1}{M} h(x)$$

The validity of this depends on the ratio of the number of sample observations M and the number of cells in the histogram Q=N

This is true for vectors as well as values.

For a vector of D values $\vec{x}$ the table has D dimensions. $h(x_1, x_2, \ldots, x_D) = h(\vec{x})$

The average error depends on the ration $Q=N^D$ and M. : $E_{ms} \sim O(\frac{Q}{M})$

We need to assure that $M >> Q = N^d$

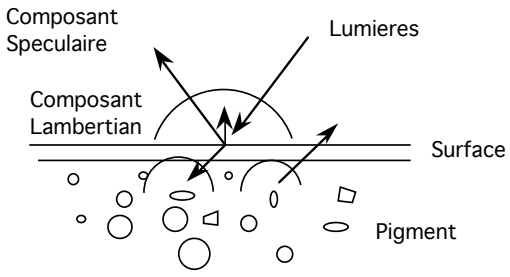
As a general rule : $M = 10N^d$

Example: Object detection by pigment color

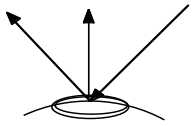
We can use Bayes rule to detect objects based on their pigment.

Recall the Bichromatic reflection function :

$$R(i, e, g, \lambda) = \alpha R_S(i, e, g, \lambda) + (1-\alpha) R_L(i, \lambda)$$



For Lambertian reflection, the intensity || P(i,j) || is generally determined by changes in surface orientation, while color is determined by Pigment.



Thus it is often convenient to transform the (RGB) color pixels into a color space that separates Luminance from Chrominance.

$$\begin{pmatrix} L \\ C_1 \\ C_2 \end{pmatrix} \Leftarrow \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Luminance captures surface orientation (3D shape)
Chrominance is a signature for object pigment (identity)

Ones such space is the color opponent model:
Luminance : L = R+G+B
Chrominance: c_1 = (R-G)/2
c_2 = B – (R+G)/2

Another, popular alternative is normalized R, G.

Luminance: L= R+G+B

Chrominance : $c_1 = r = \frac{R}{R+G+B}$ $c_2 = g = \frac{G}{R+G+B}$

Suppose that these are coded with N values between 0 and N – 1

$c_1 = \text{trunc}(N \cdot \frac{R}{R+G+B})$ $c_2 = \text{trun}(N \cdot \frac{G}{R+G+B})$

Luminance normalized RG is often used for skin detection.
Skin pigment is generally always the same color. Luminance can change with pigment density, and skin surface orientation. Chrominance will remain invariant.

Thus we can use $\vec{c} = \begin{pmatrix} r \\ g \end{pmatrix}$ as a "signature for detecting skin.

To use a Bayesian approach we need to represent the probability for each possible chrominance. We can estimate probability for chrominance with a histogram calculated from a set of training images.

Suppose the training images are composed of M pixels $\{P_m\}$.
Suppose we project these into a set of M chrominance pixels. $S = \{\vec{c}_m\}$

We then allocate a 2D table : $h(\vec{c})$ of size N x N.

For example, for skin chrominance, N=32 seems to work well.
Let $h(\vec{c})$ be a 32 x 32 table. Q = 32 x 32 = 1024 cellules

For each pixel, (i,j) possibly from S,

$$\forall_{\vec{c}_m \in S} \; h(\vec{c}_m) = h(\vec{c}_m) + 1$$

For M pixels in the training data, the histogram $h(\vec{c})$ of chrominance gives an estimate of the probability for a chrominance value within the data (or in similar data).

$$p(\vec{c}) = \frac{1}{M} h(\vec{c})$$

Important. The number of pixels, M, should be much larger than $Q = N^2$

We also can apply this to learn the probability chrominance for a target.

Suppose that we mark all pixels that belong to the target in the training data to obtain a subset $T \subset S$ composed of M_k target pixels.

We can learn a second histogram:

$$\forall_{\vec{c}_m \in T} \; h_k(\vec{c}_m) = h_k(\vec{c}_m) + 1$$

then the probability of observing a chrominance value $\vec{c}$ given the target is

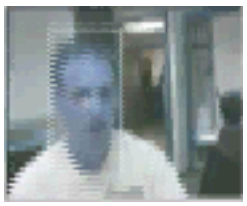
$$p(\vec{c} \mid \text{target}) = \frac{1}{M_k} h_k(\vec{c})$$

Because the target samples are a subset of the training data, the probability of a target pixel is

$$p(\text{target}) = \frac{M_k}{M}$$

From Bayes rule:

$$p(\text{target} \mid \vec{c}(i,j)) = \frac{p(\vec{c}(i,j) \mid \text{target})p(\text{target})}{p(\vec{c}(i,j))} = \frac{\frac{1}{M_k} h_k(\vec{c}(i,j)) \frac{M_k}{M}}{\frac{1}{M} h(\vec{c}(i,j))} = \frac{h_k(\vec{c}(i,j))}{h(\vec{c}(i,j))}$$



We can use this to convert a color image to a "probability image", t(i,j), by table lookup.

$$t(i,j) = \frac{h_k(\vec{c}(i,j))}{h(\vec{c}(i,j))}$$

Histograms of Receptive Field Values

This method can be generalised to ANY vector of features. For example, the appearance of a neighborhood given by the receptive field vector.

$$\vec{a}(i, j, \sigma_i) = p(i,j) * (G_x, G_y, G_{xx}, G_{xy}, G_{yy}) \text{ at some } \sigma_i$$

Given M samples of pixels $S=\{\vec{a}_m\}$ with a subset $T \subset S$ composed of M_k target pixels.

We construct a probability image for the target, t(i,j) as

$$t(i,j) = p(\text{target} \mid \vec{a}(i,j)) = \frac{h_k(\vec{a}(i,j))}{h(\vec{a}(i,j))}$$

We must, however assure that the number of sample $M_k >> Q$ the number of cells in the histogram. For D features and a quantification of N levels per feature $Q=N^D$

Gaussian Blobs

To construct a Bayesian tracker, we need to represent clouds of high target probability using Gaussian Blobs.

Gaussian blobs express a region in terms of moments.
Confidence is the sum (mass) of the detection probability pixels, t(i,j).
Position is center of gravity.

Size is the second moment (covariance).

We use some form of "a priori" estimation to estimate a Region of Interest (ROI) for the blob. Let us represent the ROI as a rectangle : (t,l,b,r)

t - "top" - first row of the ROI.
l - "left" - first column of the ROI.
b - "bottom" - last row of the ROI
r - "right" -last column of the ROI.

(t,l,b,r) can be seen as a bounding box, expressed by opposite corners (l,t), (r,b)

Moment Calculations for Blobs

Given a target probability image t(i,j) and a ROI (t,l,b,r):

Sum: $$S = \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j)$$

We can estimate the "confidence" as the average detection probability:

Confidence: $$CF = \frac{S}{(b-t)(r-l)}$$

First moments: $$\mu_i = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j) \cdot i \qquad \mu_j = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j) \cdot j$$

Position is the center of gravity: (μ_i, μ_j)

Second Moments:

$$\sigma_i^2 = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j) \cdot (i-\mu_i)^2$$

$$\sigma_j^2 = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j) \cdot (j-\mu_j)^2$$

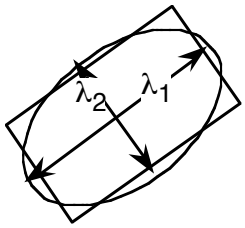
$$\sigma_{ij}^2 = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} t(i,j) \cdot (i-\mu_i) \cdot (j-\mu_j)$$

These compose the covariance matrix:

$$C = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$$

The principle components (λ_1, λ_2) determine the length and width.
The principle direction determines the orientation of the length.

We can discover these by principle components analysis.



$$\Phi C \Phi^T = \Lambda = \begin{pmatrix} \lambda_1^2 & 0 \\ 0 & \lambda_2^2 \end{pmatrix}$$

where

$$\Phi = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

The length to width ratio, λ_1/λ_2, is an invariant for shape.

This suggests a "feature vector" for the blob: $\begin{pmatrix} x \\ y \\ w \\ h \\ \theta \end{pmatrix}$

where x= μ_i, $y = \mu_j$, $w=\lambda_1$, $h=\lambda_2$

and

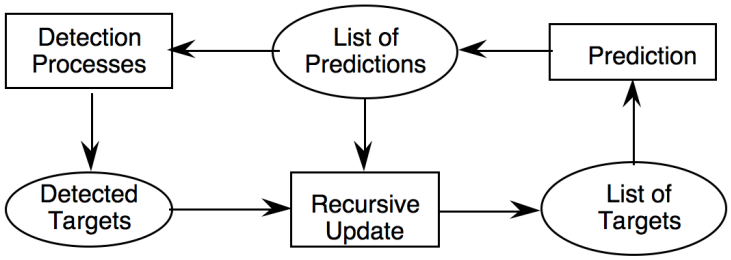
$$CF = \frac{S}{(b-t)(r-l)}$$

However, for tracking we need to keep explicit the center of gravity and covariance. Thus we will track:

Position: $\vec{\mu}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ Size : $C_t = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$ along with CF_t.

Bayesian Estimation:

A Bayesian tracker is a recursive estimator, composed of the phases:
Predict, Detect, estimate.
Having "detected a blob", next we need estimate the parameters.



The detection process can contain errors due to missed detection and false detection. To minimize the influence of errors we use the idea of a Gaussain window.

The Gaussian window is the previous covariance for the blob, enlarged by some "uncertainty" covariance. The uncertainty captures the possible loss of information during the time from the most recent observation.

Our Gaussian blob is

Position: $\vec{\mu}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ Size : $C_t = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$ along with $CF_{t.}$

Let us represent the estimated blob at time t as: $\hat{\mu}_t$, $\hat{C}_t$
Let us estimate the predicted feature vector at time t as: $\vec{\mu}_t^*$, C_t^*

We will compute the estimated blob from by multiplying the detected pixels by a Gaussian mask determined from the predicted blob. The Covariance is multiplied by 2 to offset the fact that we will use mask to estimate a new covariance.

Gaussian Mask: $G(\vec{\mu}_t^*, 2C_t^*)$

Detected target pixels:

$$t(i,j) \leftarrow \frac{h_t(c(i,j))}{h(c(i,j))} \cdot e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)^T 2C_t^{*-1} \left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_t \\ \mu_t \end{pmatrix}\right)}$$

We then estimate the new position and covariance as before:

First moments: $\mu_i = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} t(i,j)\cdot i \qquad \mu_j = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} t(i,j)\cdot j$

Second Moments:

$$\sigma_i^2 = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} t(i,j)\cdot (i-\mu_i)^2$$

$$\sigma_j^2 = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} t(i,j)\cdot (j-\mu_j)^2$$

$$\sigma_{ij}^2 = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} t(i,j)\cdot (i-\mu_i)\cdot (j-\mu_j)$$

Position: $\hat{\vec{\mu}}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ Size : $\hat{C}_t = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$

Temporal Prediction

The scene evolves. Targets move.

In the absence of a temporal model, we can estimate the vector at time t from a vector at t–Δt. We will call this an order zero model.

$$\vec{\mu}_t^* \leftarrow \hat{\mu}_{t-\Delta t}$$

If we have a temporal model, we can estimate

$$\vec{\mu}_t^* \leftarrow \hat{\mu}_{t-\Delta t} + \Delta t \cdot \frac{d\hat{\mu}_{t-\Delta t}}{dt}$$

To account for loss in precision of the blob size and position, we add a covariance

$$C_t^* = \hat{C}_{t-\Delta t} + Q_{\Delta t}$$

If we have a temporal model, we can also include this. We will see this in a minute.

We then use the predicted position and size to compute a new predicted ROI for the next time step.

Managing Lost Targets

Targets can disappear due to occlusion or lost tracking. For stability we accumulate confidence of targets over time.

$$\mathrm{CF_t} = \mathrm{CF_{t-\Delta t}} + \frac{S}{(b-t)(r-l)} - \mathrm{CF_{min}}$$

if $\mathrm{CF_t} > \mathrm{CF_{max}}$ then $\mathrm{CF_t} := \mathrm{CF_{max}}$

$\mathrm{CF_{min}}$ is the minimum required average probability per pixel to detect a target.

If $\mathrm{CF_t} \le 0$ then a target is removed.

Detecting New Targets

Where do targets come from? New detection.

Attempting to detect new targets at all positions can be expensive and can lead to errors. Two alternative techniques canbe used:

1) Trigger regions: In many scenes, new targets always appear in the same parts of the image. Trigger regions (ROI's) canbe placed at these position. In each cycle the process evaluates possible detection at some number of Trigger ROIs. If a target is detected it is added to the list.

A Trigger ROIs should now be evaluated if it overlaps an existing target.

A trigger ROI should be slightly larger than an expected target.

The number of Trigger ROIs evaluated can be varied in each cycle to assure real time response.

2) Stochastic Detection: If the position of target entry can not be predicted, then it is possible to use stochastic detection. In Stochastic detection, a few Trigger ROIs are evaluated in each cycle at random positions. As before trigger Rois should not overlap with existing targets.

SPLIT and Merge.

A classic hard problem with Bayesian Tracking is target split and target merge.

Target split happens when a single target divides into two separate parts. If the parts diverge, and the tracker follows both parts, then average confidence rapidly goes to zero. A solution is to detect the split and create a new target for each part. This requires a split detection process.

One solution is when average detection confidence is low, attempt do compute detection confidence in subregions.

If two targets come together to occupy the same pixels (as with occlusion), we have the problem of target merge. One solution is to continue tracking both targets at the same location. An alternate is to combine the targets.