Computer Vision

James L. Crowley

M2R MoSIG option GVR Fall Semester
17 October 2013

Lesson 4

Detecting and Tracking Objects with Color

<u>Lesson Outline</u>:

1 Detection and Tracking using Color

1.1 Histograms as an Estimation of Probability of Color

A histogram is a table of frequency of occurrence. Histograms have many uses in image processing and computer vision. One such use is for pixel level probabilistic detection of objects based on local appearance. A simple example is the use of color histograms to detect objects based on color statistics.

Assume that we have a color image, where each pixel *(i,j)* is a color vector, $\vec{c}(i,j)$, composed of 3 integers between 0 and 255 representing Red, Green and Blue.

$$\vec{c}(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

We can build a color histogram of the image by counting the number of times that each unique value of (R, G, B) occurs in the image. To do this we allocate a table h(r, g, b) of 256x256x256 cells, with each cell initially set to zero. The table h(r,g,b) has $256^3 = 2^{24} =$ 16 Mega Pixels.

We then visit each pixel and add one to the value of the cell that corresponds to its value of R, G, B

$$\forall_{i,j}\, h(\vec{c}(i,j)) = h(\vec{c}(i,j)) + 1$$

The table $h(\vec{c})$ then represents the frequency of occurrence for each possible color vector $\vec{c}$. Given that the image is composed of M pixels, then the this table tells us the probability of finding a pixel of color $\vec{c}$ at any position in the image.

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

If we take a second image of the same scene, and we assume that the color and illumination are similar, we can use the histogram to predict the probability of colors vectors in the second image.

1.2 Bayesian Object Detection with Color

To use color statistics for detecting objects in images, we need to have a set of labeled "training images" in which examples of the object have been identified. This is can be done by having a human trace the boundaries around examples of the object in the training images.

Suppose that we have K RGB images composed of R·C pixels, $C_k(i, j)$. This gives a total of M=K·R·C pixels. Suppose that we have a subset, *T* (for target) of these pixels that belong to a target class. Suppose that *T* contains M_T pixels.

We allocate two tables h(r, g, b) and h_t(r, g, b) and as before and use these to construct two histograms ;

$$\underset{i,j,k}{\forall} h(\vec{c}_k(i,j)) = h(\vec{c}_k(i,j)) + 1$$

$$\underset{(i,j,k)\in T}{\forall} h_T(\vec{c}_k(i,j)) = h_T(\vec{c}_k(i,j)) + 1$$

For any color vector, $\vec{c}$, have TWO probabilities :

$$P(\vec{c}) = \frac{1}{M} h(\vec{c}) \qquad \text{and} \qquad P(\vec{c} \mid T) = \frac{1}{M_T} h_T(\vec{c})$$

Bayes rule tells us that we can estimate the probability that a pixel belongs to an object given its color as :

$$P(T \mid \vec{c}) = \frac{P(\vec{c} \mid \mathrm{T})P(\mathrm{T})}{P(\vec{c})}$$

We have $P(\vec{c} \mid \mathrm{T})$ and $P(\vec{c})$. *P(T)* is the probability that a pixel belongs to a target. For the training image, this is given by

$$P(T) = \frac{M_T}{M}$$

From this we can show that the probability of a target, T, is simply the ratio of the two tables.

$$P(T \mid \vec{c}) = \frac{P(\vec{c} \mid \mathrm{T})P(\mathrm{T})}{P(\vec{c})} = \frac{\frac{1}{M_T} h_t(\vec{c}) \cdot \frac{M_t}{M}}{\frac{1}{M} h(\vec{c})} = \frac{h_T(\vec{c})}{h(\vec{c})}$$

We can use this to compute a lookup table $L_t(\vec{c})$ $\quad L_T(\vec{c}) = \frac{h_T(\vec{c})}{h(\vec{c})}$

If we ASSUME that a new image, x(i,j), has similar illumination and color composition then we can use this technique to assign a probability to each pixel by table lookup. The result is an image in which each pixel is a probability $P_T(i,j)$ that the pixel (i,j) belongs to the target T.

$$P_T(i,j) = L_T(x(i,j))$$

The reliability is improved by using more training images.

The naive statistics view says to have at least 8 training samples for histogram cell. For example, in our RGB example, h(c) was composed of

$$Q = 2^8 \cdot 2^8 \cdot 2^8 = 2^{24} \text{ cells.}$$

Thus we need $M = 2^3 \cdot 2^{24} = 2^{27}$ training pixels. This is not a problem for $P(\vec{c}) = \frac{1}{M} h(\vec{c})$ but may be a problem for $P(\vec{c} \mid T) = \frac{1}{M_T} h_T(\vec{c})$.

(Note that a 512 x 512 image contains 2^{20} pixels.)

$Q = N^D$ where N is the number of values and D is the number of features.

A more realistic view is that the training data must contain a variety of training samples that reflect that variations in the real world.

What can we do? Often we can reduce both the number, D, of features and the number of values, N, for each feature.

For example, for many color images, N=32 color values are sufficient to detect objects. We simply divide each color R, G, B by 8.

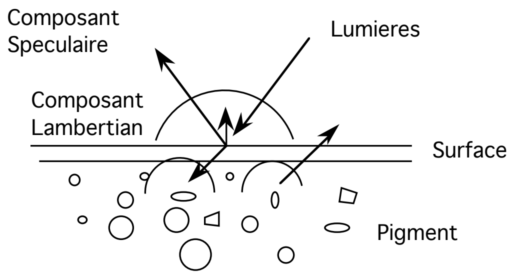
R' = Trunc(R/8), G'=Trunc(G/8), B'=Trunc(B/8).

We can also use our knowledge of physics to look for features that are "invariant".

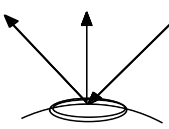
1.3 Object detection by pigment color

Recall the Bichromatic reflection function:

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



For Lambertien reflection, the intensity is generally determined by surface orientation, while color is determined by Pigment.



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

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

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

A popular space for skin detection is normalized R, G.

Luminance: $L = R+G+B$

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

These are often called "r" and "g" in the literature.

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

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

Skin pigment is generally always the same chrominance value. Luminance can change with pigment density, and skin surface orientation, but chrominance will remain invariant.

Thus we can use $\begin{pmatrix} r \\ g \end{pmatrix}$ as an invariant color signature for detecting skin in images.

1.4 Color Skin Detection

Suppose we have a set of K training images $\{c_k(i,j)\}$ of size RxC where each pixel is an RGB color vector. This gives a total of M=KxRxC color pixels. Suppose that M_{Skin} of these are labeled as skin pixels.

We can simplify our technique by projecting these onto Chrominance pixels. From experience, N = 32 color values seems to work well for skin.

We allocate two table : *h(r,g)* and *h_{skin}(r,g)* of size N x N.

```
For all i,j,k in the training set {c_k(i,j)} :
BEGIN
    r = $trunc( N \cdot \frac{R}{R+G+B})$        g = $trunc(N \cdot \frac{G}{R+G+B})$

    $h(r,g) = h(r,g)+1$

    IF the pixel P_k(i,j) is skin THEN
        $h_{skin}(r,g) = h_{skin}(r,g)+1$

END
```

As before, we can obtain a lookup table $L_{skin}(r,g)$ that gives the probability that a pixel is skin.

$$L_{skin}(r,g) = \frac{h_{skin}(r,g)}{h(r,g)}$$

Given a new RGB image C(i,j):

$$r = trunc(N \cdot \frac{R}{R+G+B}) \qquad g = trunc(N \cdot \frac{G}{R+G+B})$$

$P_{skin}(i,j) = L_{skin}(r(i,j), g(i,j))$

(example of Bayesian skin tracking in real time - 1993)

We can improve the robustness and performance of this by tracking skin regions.

2 Bayesian Tracking of Gaussian Blobs

Rather than represent a skin region as a collection of pixels, we can calculate a Gaussian Blob. A "Blob" represents a region of an image. Gaussian blobs express a region in terms of moments.

Assume of image of probabilities of the detection of a target: $P_T(i,j)$

The zeroth moment of the probabilities is the mass (sum of probabilities). Average mass represents confidence.

The first moment gives is the center of gravity. This is the "position" of the blob.

The second moment is the covariance. This gives size and orientation.

We typically enclose the blob in some rectangular Region of Interest (ROI) in order to avoid "distraction" by neighboring blobs.

The ROI is obtained by some form of estimation or a priori knowledge.

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)
We will compute the moments within this ROI (bounding box).

2.1 Moment Calculations for Blobs

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

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

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

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

First moments:

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

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

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

Second Moments:

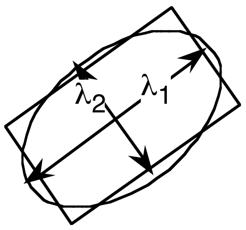
$$\sigma_i^2 = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} P_T(i,j)\cdot(i-\mu_i)^2$$

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

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

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.



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

where

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

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 = μ_j, w=λ_1, h=λ_2

and

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

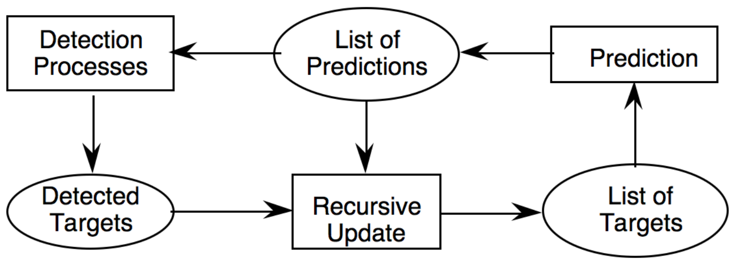
However, for tracking we need to keep explicit the center of gravity and covariance. Thus we will track:

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 $CF_{t.}$

2.2 Bayesian Tracking

A Bayesian tracker is a recursive estimator, composed of three phases:

Predict, Detect, Update.



Detection can be provided the blob using a ratio of histograms.
The following describes prediction and estimation (updating).

Our Gaussian blob is $B(\vec{\mu}_t, C_t, CF_t)$ represented by

Position: $\vec{\mu}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$

Size : $C_t = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$

Confidence: CF_t.

We will represent the estimated blob at time t as: $B(\hat{\mu}_t,\ \hat{C}_t,\ CF_t)$
We will represent the predicted blob at time t as: $B(\vec{\mu}_t^*,\ C_t^*,\ CF_t)$

2.3 Temporal Prediction

The scene evolves. Targets move. If we have an estimate of motion as

$$\frac{d\hat{\mu}_{t-\Delta t}}{dt} = \begin{pmatrix} \frac{d\mu_i}{dt} \\ \frac{d\mu_j}{dt} \end{pmatrix}$$

then from the time step, ΔT we can predict the new position:

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

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

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

To account for loss of position and size due to motion we will increase the covariance by an estimated "error" matrix, $Q_{\Delta t}$:

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

The values in $Q_{\Delta t}$ can be "calibrated" by measuring the average error between predicted and observed blobs from a labeled training sequence, or it can be "estimated".

This gives a predicted target blob represented by a Gaussian (or Normal) function:

$$\mathcal{N}(\vec{X}; \vec{\mu}_t^*, C_t^*) = \frac{1}{(2\pi)^{\frac{D}{2}} \det(C_t^*)^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{X}-\vec{\mu}_t^*)^T C_t^{*-1} (\vec{X}-\vec{\mu}_t^*)} \quad \text{where } \vec{X} = \begin{pmatrix} i \\ j \end{pmatrix}$$

We then use the predicted target blob to compute a new predicted ROI for the detection as described above.

2.4 Updating the Estimated Blob Parameters

Once we have detected the blob, we must update the estimation of its parameters. Detection can introduce new errors, such distraction by adjacent target
To minimize such errors, we use a technique from robust estimation.
We weight the detected pixels by the predicted Normal density.

$$\hat{P}(i,j) = P_T(i,j) \cdot \mathcal{N}(i,j; \vec{\mu}_t^*, C_t^*)$$

We then estimate the new position and covariance using this product:

First moments: $\hat{\mu}_i = \frac{1}{S}\sum_{i=l}^{r} \sum_{j=t}^{b} \hat{P}(i,j) \cdot i \qquad \hat{\mu}_j = \frac{1}{S}\sum_{i=l}^{r} \sum_{j=t}^{b} \hat{P}(i,j) \cdot j$

Second Moments:

$$\hat{\sigma}_i^2 = \frac{1}{S}\sum_{i=l}^{r} \sum_{j=t}^{b} \hat{P}(i,j) \cdot (i-\hat{\mu}_i)^2$$

$$\hat{\sigma}_j^2 = \frac{1}{S}\sum_{i=l}^{r} \sum_{j=t}^{b} \hat{P}(i,j) \cdot (j-\hat{\mu}_j)^2$$

$$\hat{\sigma}_{ij}^2 = \frac{1}{S}\sum_{i=l}^{r} \sum_{j=t}^{b} \hat{P}(i,j) \cdot (i-\hat{\mu}_i) \cdot (j-\hat{\mu}_j)$$

which gives: $\hat{\vec{\mu}}_t = \begin{pmatrix} \hat{\mu}_i \\ \hat{\mu}_j \end{pmatrix}$ and $\hat{C}_t = \begin{pmatrix} \hat{\sigma}_i^2 & \hat{\sigma}_{ij}^2 \\ \hat{\sigma}_{ij}^2 & \hat{\sigma}_j^2 \end{pmatrix}$

2.5 Managing Lost Targets

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

$$CF_t = \alpha\ CF + (1- \alpha)\ CF_{t-\Delta t} +$$

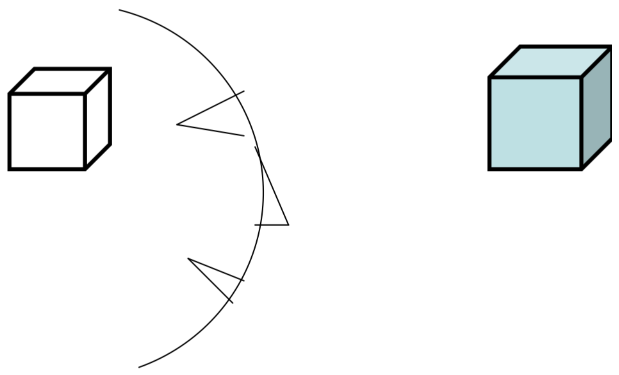
CF_{min} is the minimum required average probability per pixel to detect a target.

If $CF_t \leq CF_{min}$ then a target is removed.

The weight α determines the decay rate for lost targets.

3 Describing Images with Derivatives

Objects composed of planar surfaces (polyhedric objects) have straight line edges. Such edges are visual invariants to view angle. For this reason, edges (straight edge contours) became popular as an invariant image description during the 1960s and 1970s.



Edge detection is typically organized in two steps
1) contrast filtering
2) edge point detection, segmentation and description.

$$\begin{tikzcd}\text{image} \arrow[r] & \boxed{\text{Filter}} \arrow[r] & \text{image} \arrow[r] & \boxed{\text{Detection}} \arrow[r] & \text{Edge Description}\end{tikzcd}$$

A classic contrast detection operator is the Sobel edge detector.
Modern approaches use Gaussian Derivatives.

3.1 The Sobel Edge Detector

Invented by Irwin Sobel in his 1964 Doctoral thesis, this edge detector was made famous by the classic text book of R. Duda and P. Hart published in 1972.

$$m_r(i,j) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad m_c(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$m_c(i,j)$: detects contrast in column direction. *$m_r(i,j)$* : detects contrast in row direction

Convolution (or filtering) gives an edge vector: for n = r, c (row or column)

$$E_n(i,j) = m_n * p(i,j) \sum_{k=-1}^{1}\sum_{l=-1}^{1} m_n(k,l)p(i-k,j-l) \qquad \vec{E}(i,j) = \begin{pmatrix} E_c(i,j) \\ E_r(i,j) \end{pmatrix} = \begin{pmatrix} m_c(i,j) \\ m_r(i,j) \end{pmatrix}$$

(here we introduce the subscript as a derivative notation).

The contrast is the edge energy: $E(i,j) = \|\vec{E}(i,j)\| = \sqrt{E_r(i,j)^2 + E_c(i,j)^2}$

The direction of maximum contrast is the phase angle $\varphi(i,j) = Tan^{-1}\left(\frac{E_c(i,j)}{E_r(i,j)}\right)$

Sobel's edge filters can be seen as a composition of an image derivative and a smoothing filter.

$$m_c(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} = b_c(i,j) * d_r(i,j) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} b_c(k,l)d_r(i-k,j-l)$$

$$m_r(i,j) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} = b_r(i,j) * d_c(i,j) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} b_r(k,l)d_c(i-k,j-l)$$

The filter $d_c(i,j) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$ is a form of image derivative in the column direction

The filter $b_c(i,j) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ is a binomial smoothing filter in the column direction

3.2 Difference Operators: Derivatives for Sampled Signals

How can we take a derivative for a sampled signal?

For continuous x, the derivative of the function s(x) can be defined as a one sided derivative or a two sided derivative:

One sided derivative: $\frac{\partial s(x)}{\partial x} = \lim_{\Delta x \to 0} \left\{ \frac{s(x + \Delta x) - s(x)}{\Delta x} \right\}$

For a sampled signal, s(n), an the equivalent is $\frac{\Delta s(n)}{\Delta n}$

A two sided (symmetric) derivative is $\frac{\partial s(x)}{\partial x} = \lim_{\Delta x \to 0} \left\{ \frac{s(x+\Delta x) - s(x-\Delta x)}{2\Delta x} \right\}$

For a sampled signal, the two sided derivative is :

$$\frac{\Delta s(n)}{\Delta n} = \frac{s(n+1) - s(n-1)}{2} = s(n) * \left[-1/2 \quad 0 \quad 1/2 \right]$$

The ½ term is a scalar multiple that can be neglected or normalize away.
The result is the edge operator used by Sobel.

Δn = 1 : $\frac{\Delta s(n)}{\Delta n} = s(n) * \left[-1 \quad 0 \quad 1 \right]$