Computer Vision

James L. Crowley

M2R MoSIG Fall Semester
GVR and UIS 12 Oct 2017

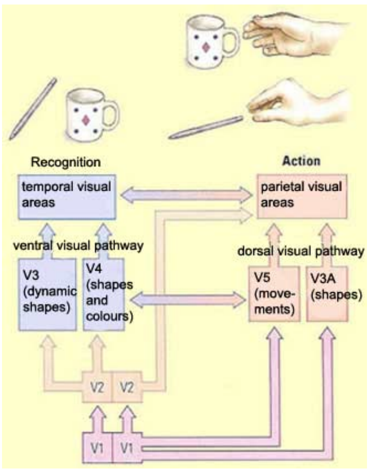
Lesson 2

Attention, Detection and Tracking

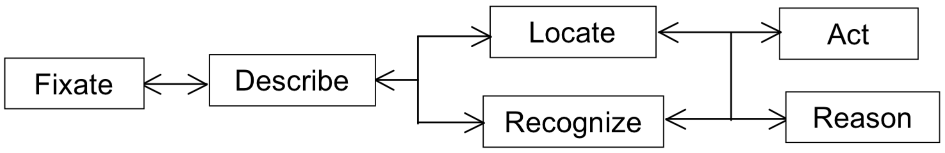
<u>Lesson Outline</u>:

1 Vision as Process

As we saw last week, human vision is performed by a series of visual pathways.



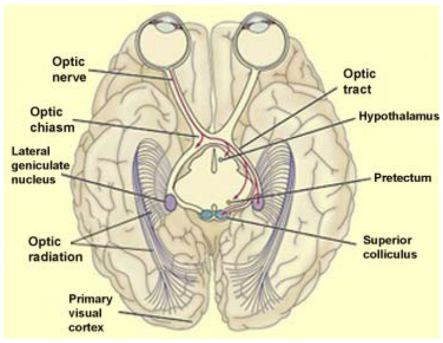
We can approximate this as a process with the following steps.

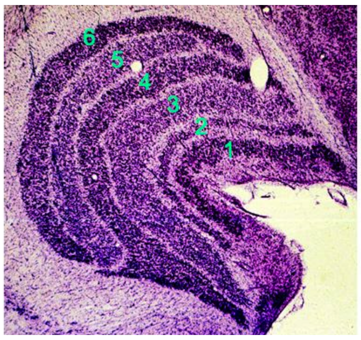


Note that each step is a tightly coupled perception-action loop.

1.1 Fixation in Human Vision

As we saw last week, fixation is provided by the Superior Colliculus (SC) and the Lateral Geniculate Nucleus (LGN).





Both the SC and the LGN are a series of layers organized as retinotopic maps, preserving the relative spatial position of information.

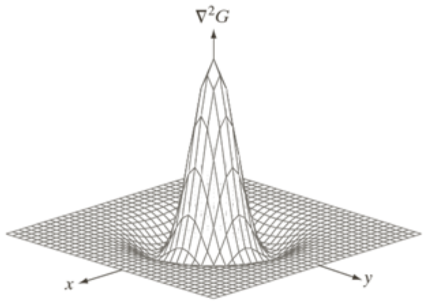
The Superior Colliculus fixates the horopter on a visual target in the scene, filtering out adjacent stimulus.

The Superior Colliculus has 7 layers from different parts of the brain and transforms signals from retinotopic coordinates to motor coordinates for the eye muscles.

The Lateral Geniculate Nucleus filters the visual pathway by suppressing non-attended information.

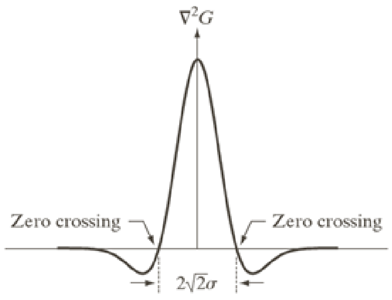
The Lateral Geniculate Nucleus has 6 layers with alternate inputs from each of the two eyes and acts to suppress non-attended visual stimuli. Detection and suppression are provided by Center Surround cells, sometimes referred to as a Mexican Hat and modeled as the Laplacian (2nd derivative) of a Gaussian function.

$$\nabla^2 G(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial \sigma}$$



2D Plot of $\nabla^2 G(x,y,\sigma)$

Image of $\nabla^2 G(x,y,\sigma)$



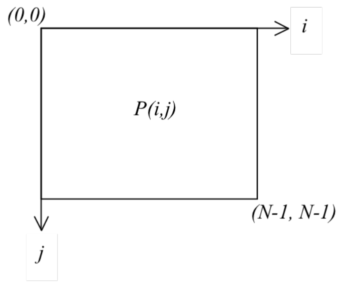
1-D Cross section $\nabla^2 G(x,y,\sigma)$

The center surround cells are a kind of “spot detector” for blobs of a particular size.

Fixation serves to reduce computational load. Without fixation, the visual cortex would require a mass of several tens of kilograms. In many cases, fixation is a learned skill. The eye-scan used in reading is a good example.

1.2 Fixation and Attention in Machine Vision

Assume an image, *P(i,j)*, in which each pixel is an 8 bit luminance value.
Let i refer to columns from 0 to N-1, and j refer to rows from 0 to N-1. A typical value of N would be 1024.

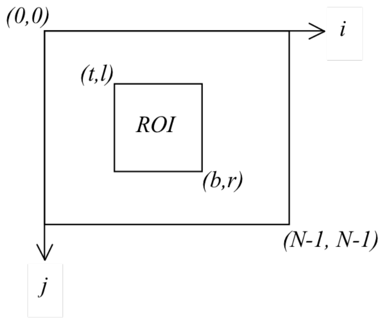


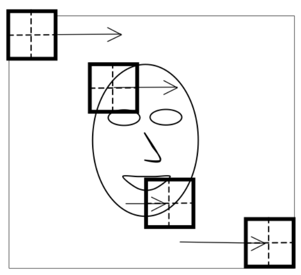
In machine vision, fixation serves to reduce computational load and reduce errors by focusing processing on parts of the image that are most likely to contain information. The fixated region is generally referred to as a "Region of Interests" (ROI). The ROI is determined either by a-prior knowledge or by some form of search procedure. In many cases this is based on tracking of targets in the scene.

The most common form of ROI is a rectangle represented by four coordinates:
(top, left, bottom ,right) or (*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).
In some vision techniques the ROI is simply a sliding window that scans the entire image. This is the technique commonly used with the Viola Jones Face Detector.





In some systems both the size and the position of the ROI are scanned. This can be on a linear scale or a logarithmic scale. Of course, with sufficient computing power, all windows can be processed in parallel.

1.3 Robust Fixation: Gaussian windows.

The problem with a rectangular window is that the window may only partially overlap the target and may include other target. For robust detection within the ROI we can use a Gaussian Window.

Gaussian Window: $G(i,j;\vec{X}) = e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)^T \Sigma^{-1} \left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)}$

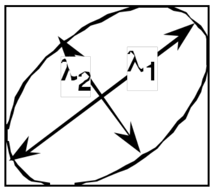
The Guassian window is an elliptical form. The center is determined by $\vec{\mu}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ the center of Gravity. The width, height and orientation are determined by a covariance matrix $\Sigma = \begin{pmatrix} \sigma_i^2 & \sigma_{ij} \\ \sigma_{ij} & \sigma_j^2 \end{pmatrix}$.

The orientation of the ellipse is determined from the eigenvectors of Σ.
The window has a value of 1.0 at the center of gravity $\vec{\mu}_t = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$, and gradually reduces to approximately zero beyond a distance of 3σ.

The width, height and orientation are determined from the principle components and principal vectors of the blob. (λ_1, λ_2, θ).
We can discover these by principle components analysis described below.

The Gaussian is typically bounded by a 3σ ROI determined from the eigenvalues:

$$
\begin{aligned}
l &= \mu_i - 3\cdot(\lambda_1 \cos(\theta) + \lambda_2 \sin(\theta)) \\
r &= \mu_i + 3\cdot(\lambda_1 \cos(\theta) + \lambda_2 \sin(\theta)) \\
t &= \mu_j - 3\cdot(-\lambda_1 \sin(\theta) + \lambda_2 \cos(\theta)) \\
b &= \mu_j + 3\cdot(-\lambda_1 \sin(\theta) + \lambda_2 \cos(\theta))
\end{aligned}
$$



A target in a Gaussian window is sometimes referred to as a Gaussian "Blob".

Target pixels *T(i,j)* are detected by multiplying the image *P(i,j)* within the fixation window within a region of interest.

$$T(i,j) \leftarrow P(i,j)\cdot G(i,j;\vec{X})$$

within the ROI.

Gaussian blobs express the spatial extent of the target in terms of moments. The zeroth moment is sum of detection energy in the target. This is equivalent to the mass in physics and can be used to estimate a confidence factor (CF) that a target has been detected within the ROI.

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

The second moment is the covariance. This gives width, height and orientation.

Normalized Gaussian Fixation Window.

In some cases it is convenient to normalize the Gausian to sum to 1.

Normalized Gaussian Window: $G(i,j;\vec{X}) = \frac{1}{2\pi \det(\Sigma)^{1/2}} e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)^T \Sigma^{-1} \left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)}$

Where : $\det(\Sigma)$ is the determinant of the covariance, Σ. $\det(\Sigma) = \sigma_i^2 \sigma_j^2 - \sigma_{ij}^2$

The term $\frac{1}{2\pi \det(\Sigma)^{1/2}}$ assures that the window weights sum to 1.

1.4 Moment Calculations for Blobs

We can initialize a Gaussian blob from the moments of detection energy inside a ROI.

Given a ROI *(r,l,t,b)* within a target detection image *T(i,j)*

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

We can estimate the "confidence" as proportional to the mass of detection probability:

Confidence: $CF = \frac{1}{\det(\Sigma)^{1/2}} S$

The term $\det(\Sigma)^{1/2}$ normalizes for the mass of the Gaussian window.
The confidence (CF) can be seen as the “Likelihood” that the model for the blob has been detected.

First moment or Center of Gravity:

The first moment is the center of gravity of the target

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

This gives a position vector *(μ_i, μ_j)*
We will use this as the position of the blob.

Scale and orientation or Second Moments:

The second moments tell the spatial extent of the blob.

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

These compose a covariance matrix: $\Sigma = \begin{pmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{pmatrix}$

This suggests a "feature vector" for the blob: $\vec{X}_n = \begin{pmatrix} \mu_i \\ \mu_j \\ \sigma_i^2 \\ \sigma_j^2 \\ \sigma_{ij} \end{pmatrix}$

We can use the feature vector to determine position, width, height and orientation :

$$\vec{X} = \begin{pmatrix} c_i \\ c_j \\ w \\ h \\ \theta \end{pmatrix}$$

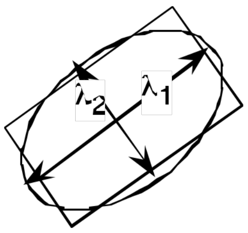
where (c_i, c_j) is the center position, and width, height and orientation are determined from the principle components and principal vectors of the blob.

(λ_1, λ_2, θ).

We can discover these by principle components analysis.

1.5 Principle Components Analysis

The principle components of the blob are found by rotating the covariance to for a diagonal matrix.



$$RCR^T = \Lambda = \begin{pmatrix} \lambda_1^2 & 0 \\ 0 & \lambda_2^2 \end{pmatrix} \qquad \text{where} \qquad R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

The principle components, λ_1, λ_2, are the "eigenvalues" or characteristic values of Σ.

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

The angle θ is a "Covariant" for orientation.

We can use λ_1 and λ_2, to define the "width and height" of the blob:

Width: $h = 2\lambda_1 + 1$
Heigth: $w = 2\lambda_2 + 1$

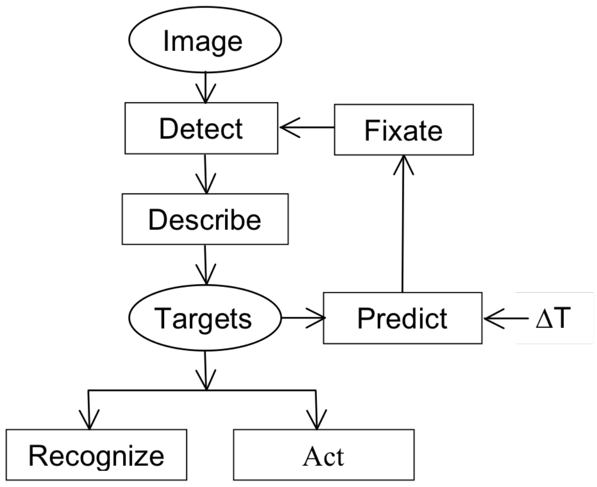
where $c_i = \mu_i$, $c_j = \mu_j$, $w = \lambda_1$, $h = \lambda_2$ and $\theta = \tan^{-1}\left(\frac{r_{21}}{r_{11}}\right) = \tan^{-1}\left(\frac{\sin(\theta)}{\cos(\theta)}\right)$

Tracking allows us to continually update an estimate for the features of the Blob, even if the blob is temporarily lost to occlusion or noise.

The tracked object is often referred to as a "target".

2 Tracking and Object Constancy

Tracking is a process of recursive estimation of target parameters from observations. Tracking is widely used in machine vision to provide reduce computational load, reduce errors and provide object constancy (preserve identity of entities over time).



The prediction phase compensate for unknown motions by making the Gaussian fixation window larger. This can be done by adding a second covariant matrix:

$$\Sigma_{t+\Delta t} \leftarrow \Sigma_t + Q\Delta t \quad \text{where } Q = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}$$

Note that the smaller ΔT, the smaller the resulting ROI.
It is also possible to maintain an estimate of velocity and acceleration of targets and to update the estimated position during prediction.

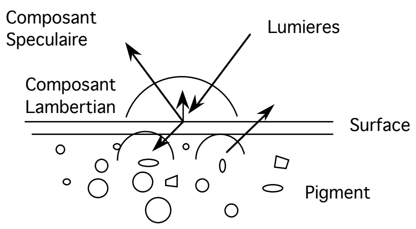
We can use many different methods to detect the pixels of target entities.

2.1 Detection using Ratio of Color Histograms

Assume a color image, $C(i,j)$, where each pixel is three 8 bit values for R, G and B.

Recall the Bidirectional Reflectance Distribution Function (BRDF) tells us that most objects can be modeled as a sum of a lambertian and specular reflection. :

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



For Lambertian reflection, the intensity is determined by surface orientation, while color is determined by pigment. Thus we can use pigment color acts as a signature for the object. We can easily this signature to determine the probability of a target at each pixel. This can be done with table lookup using a ratio of histograms.

<u>Representing Probability with histograms</u>

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, $C(i,j)$, composed of 3 integers between 0 and 255 representing Red, Green and Blue.

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

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 256 x 256 x 256 cells, with each cell initially set to zero.
The table *h(r, g, b)* has $256^3 = 2^{24} =$ 16 M cells.

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

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

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

$$P(C)(i,j) = \frac{1}{M} h(C(i,j))$$

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.

<u>Bayesian Object Detection with Color</u>

Suppose that we have K 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.

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

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

For the color vector, $\vec{c} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$ at any pixel $C(i,j)$ 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 any pixel belongs to a target. This can be estimated from the training data by:

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

From this we can show that the probability of a target, T, at pixel *(i,j)* 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 $L_T(\vec{c})$: $L_T(\vec{c}) = \frac{h_T(\vec{c})}{h(\vec{c})}$

If we ASSUME that a new image, C(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(i,j)* that the pixel (i,j) belongs to the target T.

$$P(i,j) = L_T(C(i,j))$$

The reliability is improved by using training images with diverse lighting conditions.

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

$Q = 2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ 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 1024 x 1024 image contains 2^{20} pixels. This is the definition of 1 Mega)

Q is the "capacity" of the histogram, measured as the number of cells.

$Q = N^D$ where N is the number of values per feature 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".

<u>Color Skin Detection</u>

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.

$$\begin{pmatrix} L \\ c_1 \\ c_2 \end{pmatrix} \Leftarrow \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Normalizing out luminance provides a popular space for skin detection: the (r,g) space.

Luminance: L= R+G+B

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

These are often called "r" and "g" in the literature.
The *(r, g)* space is often used to detect skin colored pixels.

r and g have values between 0 and 1.
It is common to represent r and g as natural numbers coded with N values between 0 and N – 1 by :

$$r = trunc\left((N-1)\cdot\frac{R}{R+G+B}\right) \quad g = trunc\left((N-1)\cdot\frac{G}{R+G+B}\right)$$

The color of pigment for any individual is generally constant. Luminance can change with pigment density (eg. Lips), and skin surface orientation, but chrominance will remain invariant.

From experience, N = 32 color values seems to work well for skin with most web cameras.

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

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.
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
        $r = trunc\left((N-1)\cdot\frac{R}{R+G+B}\right)$  $g = trunc\left((N-1)\cdot\frac{G}{R+G+B}\right)$

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

        IF the pixel $c_k(i,j)$ 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):

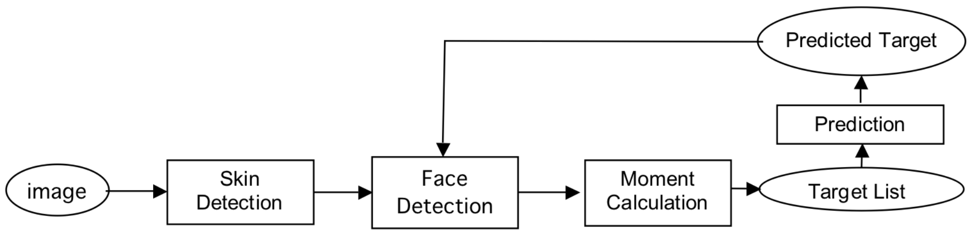
$$r = trunc\left((N-1)\cdot\frac{R}{R+G+B}\right) \quad g = trunc\left((N-1)\cdot\frac{G}{R+G+B}\right)$$

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

(images from a Bayesian skin tracking in real time - 1993)

2.2 Robust Tracking of Skin Blobs

We can improve the detection by robust tracking of skin colored regions.



The algorithm maintains a list of targets, X_n.

Target parameters are $\vec{X}_n = \begin{pmatrix} \mu_x \\ \mu_y \\ \sigma_i^2 \\ \sigma_j^2 \\ \sigma_{ij} \end{pmatrix}$ (for N targets).

Each target is fixated with a Gaussian Fixation window:

$$G(i,j;\vec{X}) = e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)^T 2\Sigma^{-1}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)}$$

This is a form of robust estimation that uses the Gaussian Fixation to reject outliers.

Assume a color image : $C(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}(i,j)$ and a ROI window *(t,b,l,r)* for detection.

The algorithm uses a lookup table for ratio of histograms to compute an image of the probability of skin. The lookup table gives a probability for each color value:

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

1) Color Skin detection:
For each pixel, *C(i,j)*, compute normalized color *r,g*

$$r = trunc\left((N-1)\cdot\frac{R}{R+G+B}\right) \quad g = trunc\left((N-1)\cdot\frac{G}{R+G+B}\right)$$

Use *(r,g)* to compute probability of skin at *(i,j)*

$$P(i,j) \leftarrow L(r,g)$$

2) Fixation: For each target N:

Fixated target image: $T_n(i,j) = P(i,j) \cdot G(i,j;X_n)$

3) Update moments for the target *n*.

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

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

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

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

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

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

$$T_n \leftarrow (\mu_i, \mu_j, \sigma_i, \sigma_j, \sigma_{ij})^T$$

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

4) Suppress target from detected probability image:

$$P(i,j) \leftarrow P(i,j)\big(1 - G(i,j;X_n)\big)$$

Where G is the unnormalized Gaussian window (max of 1 at center).

5) Detect any new targets by fixating at an initialization windows. (depends on application).
Note that the confidence is proportional to the average probability within the ROI.
We can use the confidence to determine if a blob was detected.

For an initial detection we initiate a new target if the CF is above a threshold.

During tracking we can use a recursive decay to average the CF over the last frames.
This prevents the algorithm from dropping targets that are temporarily occluded.

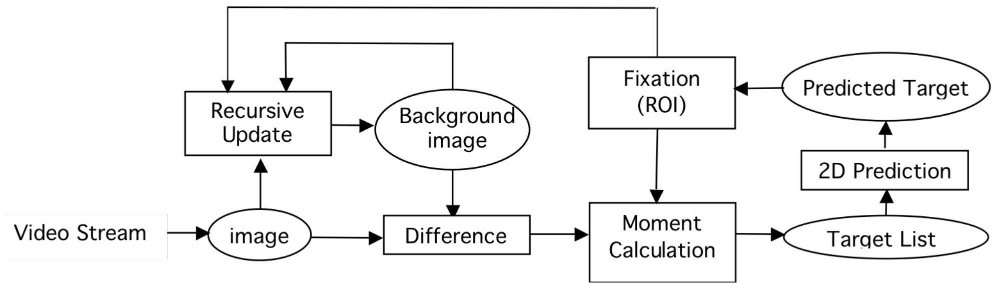
Let CF_t be the average confidence at time t. we update with:

$$CF_t \leftarrow aCF + (1-a)CF_{t-1}$$

where a is a small value (for example 0.1).

2.3 Detection by Background Difference Subtraction

Background Difference Subtraction is a widely used for Video Surveillance. This algorithm assumes that the camera is stationary.



Tuned Parameter: α update rate. (Typically ~ 0.01)
Algorithm uses a Gaussian Blob for each target.

Target parameters $\vec{X}_n = \begin{pmatrix} \mu_x \\ \mu_y \\ \sigma^2_i \\ \sigma^2_j \\ \sigma_{ij} \end{pmatrix}$ (for N targets).

Fixates on Targets with a Gaussian Fixation window:

$$F(i,j;X) = e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)^T 2\Sigma^{-1}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \mu_i \\ \mu_i \end{pmatrix}\right)}$$

Algorithm:

0) Initialize: Acquire an empty background image
$B(i,j) \leftarrow P(i,j)$

Loop:
1) Difference from Background: $D(i,j) = P(i,j) - B(i,j)$
We subtract an average background from the latest image.
Note Difference can be positive or negative!

2) Fixation: For each target N detected in previous images

Fixated target image: $T_n(i,j) = |D(i,j)| \cdot G(i,j;X_n)$

Where G is the Gaussian fixation window from tracking/

3) Update moments for each Target, n

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

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

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

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

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

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

$$T_n = (\mu_i, \mu_j, \sigma_i, \sigma_j, \sigma_{ij})^T$$

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

$CF_t \leftarrow \beta CF + (1-\beta)CF_t$ is the recursive estimate of confidence, β is an update factor, typically 0.1.

4) Suppress target from difference image:

$$D(i,j) \leftarrow D(i,j)\big(1 - G(i,j;X_n)\big)$$

5) Detect any new targets within a detection region (algorithms vary).

6) Update background image with remaining difference image

$$B(i,j) \leftarrow \alpha\cdot D(i,j) + (1-\alpha)\cdot B(i,j)$$
(typical value is $\alpha = 0.01$)

Demonstration Video: This is a video from a tracker demonstrated at the first PETS 2000 workshop (Performance Evaluation for Tracking and Surveillance). The code was later comercialized by a start up BlueEye Video.)

2.4 Detecting New targets

Methods to detect new targets generally depend on the application domain.
Popular methods include

1) Entry regions. Place an ROI window of approx target size on areas where new targets can appear. (edges of the image, doors, roadways, etc).

Detect new targets from residue image. by computing the moments in each entry ROI window.

2) Stochastic detection: Place a ROI window of approx target size at random places in the image at the end of each cycle. Detect new targets by computing the moments in each entry ROI window.

3) Learned entry regions: Use frequency of detection for Stochastic detection to learn high entry regions.