Computer Vision

James L. Crowley

M2R MoSIG

Fall Semester
12 Nov 2020

Lesson 5

Fixation and Tracking for Computer Vision

Lesson Outline:

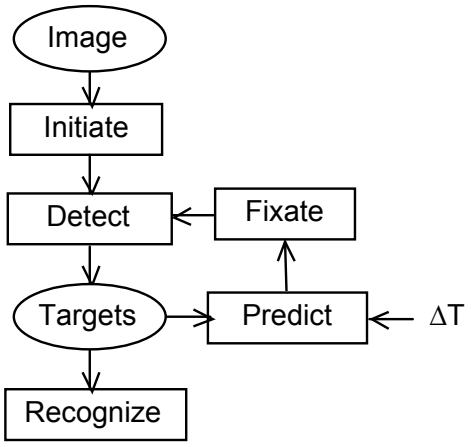
1 Fixation and Tracking

Fixation focuses processing on a particular part of the visual field. As we have seen, fixation is a key part of the human visual system. 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. In computer vision, fixation is classically provided by a tracking process, often combined with some form of model that describes what to look for and where to look next.

Tracking is a process of recursive estimation from a temporal sequence of observations. In addition to reducing computational load and reducing errors, tracking preserve the identity of observed entities over time. In cognitive vision, this is referred to as object constancy.

For historical reasons, the entities that are tracked are commonly referred to as targets. Entities are described with a set of properties, such as position, size, eccentricity, and orientation as well as rates of change (velocities) for these parameters. In a Kalman filter tracker, estimated properties are accompanied with precision and confidence, estimating the accuracy (error bounds) for each parameter, as well as the confidence for the estimate.

Starting with some initial detect, targets are tracked with a cyclic process in which a processing is focused (fixated) using estimated values for position, scale and other parameters. If a detection process confirms the presence of the predicted target, estimates for the target properties are updated from the observation. If the presence of the target is not confirmed, estimates for properties are based only on the previous observations, and the confidence of the target is decreased. When the confidence drops below a threshold, it is removed from the target list. This process protects tracking from temporary loss of targets due to noisy observations or occlusions by other targets.



1.1 Techniques for Target Detection

Many different methods have been developed to detect and track target entities. Theses include

1) Adaptive Background Subtraction followed by Bayesian tracking.
2) Color Histogram based tracking followed by Bayesian tracking.
3) SIFT features detected with Laplacian Keypoints
4) Bayesian tracking using Viola Jones Cascade detectors.
5) Bayesian tracking using Multi-Layer Perceptrons or CNNs.

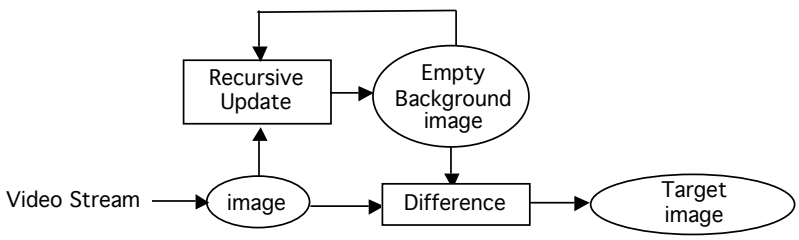
Classically, video surveillance systems use adaptive background subtraction, followed by a Bayesian estimation process that estimates the position and size of targets as Gaussian Blobs, using moments. Color histograms can also be used with Bayes rule to develop a fast process for detecting Gaussian Blobs using moments. This can be very useful for tracking faces and hands. The tracking process for Gaussian Blobs can be formalized as a Bayesian Estimation process using a form of Kalman Filter. Classic techniques for Bayesian tracking can be combined with modern machine learning using the Viola Jones cascade detectors, Multi-layer Perceptrons or Convolutional Networks.

We will first illustrate Bayesian tracking by describing the classic approach using adaptive background subtraction. Most of the above techniques can be used with adaptive background.

1.2 Adaptive Background Subtraction.

Background Difference Subtraction is a simple and effective algorithm for detection and tracking of objects and is a widely used for Video Surveillance. This technique assumes that the camera is stationary, although it can be adapted to work with a mobile camera by estimating and removing background motion due to camera motion.

The basic algorithm looks like this:



Assume a video stream that provides a temporal sequence of images $P_t(i,j)$. The process is initialized with an image $P_0(i,j)$ of the scene in which no targets are present. This empty image is copied to the current background image $B_t(i,j)$.

$$B_0(i,j) \leftarrow P_0(i,j)$$

The current background image is then subtracted from each new image to produce a difference image. Note that the difference image is signed and may be negative. The absolute value is commonly used to avoid problems with signed images.

$$D_t(i,j) \leftarrow | P_t(i,j) - B_{t-1}(i,j)|$$

This has the effect of suppressing background information, and making any new entities salient. The new image is then also used to update the background image using a recursive formula:

$$B_t(i,j) \leftarrow \alpha P_t(i,j) - (1-\alpha)B_{t-1}(i,j)$$

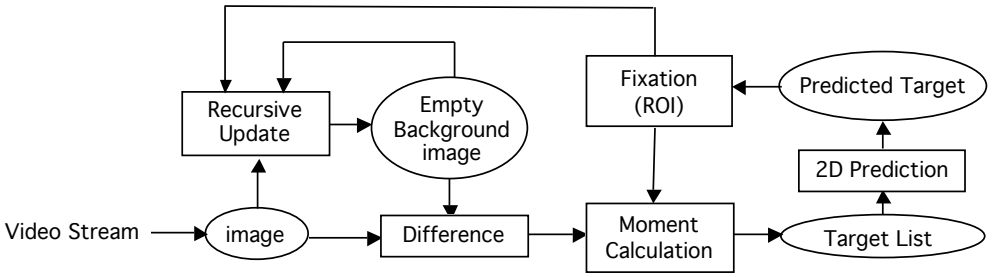
This enables the system to accommodate any changes in ambient lighting or camera motion. A typical value for α would be in the range from 0.01 to 0.001, although much larger values may be used to accommodate camera motion.

Notice that the resulting difference image is signed. Many of the image processing and display functions in popular toolboxes such as OpenCV and Scikit-Image assume unsigned images. It is may be necessary to compute an absolute value of the difference or to add 128 all pixels.

As we will see below, it is also necessary to suppress the adaptive background update process in regions where targets are detected to avoid adapting to targets.

2 Tracking Gaussian Blobs with Adaptive Background Subtraction

Adaptive background subtraction is widely used for video surveillance. A flow diagram for such a system is shown here.



Tracking is a cyclic process in which the presence of targets within a number of rectangular Region of Interests are confirmed using a simple sum of pixel energy. If the sum of the absolute value of pixels within a Region of Interest in the difference image, $D_t(i,j)$, is above a threshold then the presence of a target is confirmed. In this case, the position and extent of the target are computed using the first and second moment within the ROI, and the target properties and confidence are updated. If the pixel energy in the ROI is below the threshold then the target is considered as absent and the confidence factor for the target is reduced.

Similar to what we saw for the FDDB ground truth, targets are represented as an elliptical region, denoted by a 5-tuple (x, y, l, h, θ) where x and y refer to the column and row image coordinates of the center of this ellipse, l and h are the major and minor axes, θ is the angle of the major axis with the horizontal axis. These are commonly noted in a vector as $\vec{X}_n$.

$$\vec{X}_n = \begin{pmatrix} x \\ y \\ l \\ w \\ \theta \end{pmatrix}$$

Each target is also labeled with a confidence factor, CF, that represents the likelihood of the presence of the target. Ideally, this would be a probability, but often it is simply a numerical estimate. Generally this is based on the average energy of the target after background subtraction.

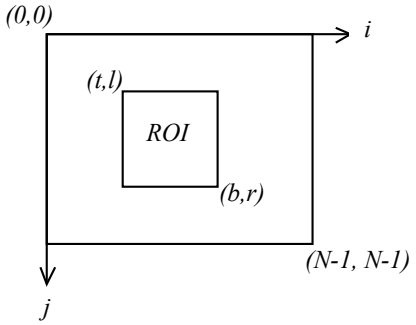
2.1 Estimating the Target Moments:

Targets are detected within a "Region of Interests" (ROI) based on previous position of targets or some other form of a-prior knowledge or search procedure.

As in earlier lectures, 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)



2.2 Detecting new targets

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

A classic technique is to identify specific regions where new targets can enter the image. This can be done by hand, or various automatic procedures that learn the entry regions form training data. For many domains, entry regions are ROIs of an average target size, placed around the boundaries of the image.

Stochastic detection works by placing a ROI of approximate target size at random non-target regions the difference image at the end of each cycle.

It is also possible to use compute a Laplacian pyramid of the difference image and then use peaks in the Laplacian as target detection regions.

In the following we will assume a small number of entry regions defined by ROIs that are permanently placed by hand within the image. Whenever the average detection energy CF for the ROI is above a threshold, a new target is created and added to the target list.

2.3 Moment Calculations for Blobs

Gaussian blobs express a 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 and provides the position of the target.The second moment is the covariance, and gives the width, height and orientation.

Given a target detection image $D_t(i,j)$ within some ROI (r,l,t,b).

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

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

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

If CF is greater than some pre-set average, Thr, then the target is judged to be present and target parameter estimation can continue:

IF CF ≥ Thr THEN Target = TRUE else Target = False

First moment or Center of Gravity:

The first moment is the center of gravity of the target

$$x = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} \mathrm{D}_t(i,j) \cdot i$$

$$y = \frac{1}{S} \sum_{i=l}^{r} \sum_{j=t}^{b} \mathrm{D}_t(i,j) \cdot j$$

This gives a position vector (x,y) for the center of the blob.

Scale and orientation or Second Moments:

The second moments tell the spatial extent of the blob.

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

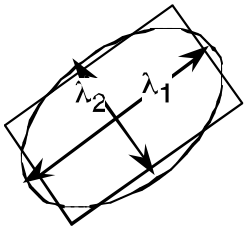
$$\sigma_j^2 = \frac{1}{S}\sum_{i=l}^{r}\sum_{j=t}^{b} \mathrm{D}_t(i,j)\cdot(j-\mu_j)^2$$

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

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

Principle Components Analysis

The length and width of the blob are determined from its principle components. These are found by determining the rotation angle that would rotate the covariance to for a diagonal matrix. This can be computed with any number of available algorithms found in most on-line toolkits for numerical computations.



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

where

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

The principle components, λ_1^2, λ_2^2, 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:

Length: $l=\lambda_1$,
Width: $w=\lambda_2$

$$\theta = \tan^{-1}\left(\frac{r_{21}}{r_{11}}\right) = \tan^{-1}\left(\frac{\sin(\theta)}{\cos(\theta)}\right)$$

Calculating the Confidence Factor.

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

$$CF_n \leftarrow \eta CF + (1-\eta)CF_n$$

4) Suppress target from difference image:

$$D_t(i,j) \leftarrow D_t(i,j)\left(1 - F_n(i,j)\right)$$

5) Detect any new targets (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 $\alpha = 0.01$)

2.4 Robust Fixation: Gaussian windows.

The problem with a rectangular window is that the window may only partially overlap the target. A more robust technique is to use a Gaussian Window. This is a form of robust estimation algorithm that gives more weight to the center of the ROI>

Gaussian Fixation Window: $F(i,j) = e^{-\frac{1}{2}\left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix}\right)^T \Sigma^{-1} \left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix}\right)}$

Where : $\hat{\mu} = \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix}$ is the predicted position of the Blob and $\Sigma = \begin{pmatrix} \sigma_i^2 & \sigma_{ij} \\ \sigma_{ij} & \sigma_j^2 \end{pmatrix}$ is the scale of the mask. The ROI is prefiltered, by multiplying the difference eimage $B_t(i,j)$ with the fixation window.

$$D_t(i,j) \leftarrow D_t(i,j) \cdot F(i,j)$$

The covariance matrix is typically chosen to fit 2 standard deviations within the ROI. IN this case, the center of the Gaussian Fixation window the coefficients is multiplied by 1 and the window reduce to 0.05 at the boundary of the ROI.

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 commercialized by a start up BlueEye Video.

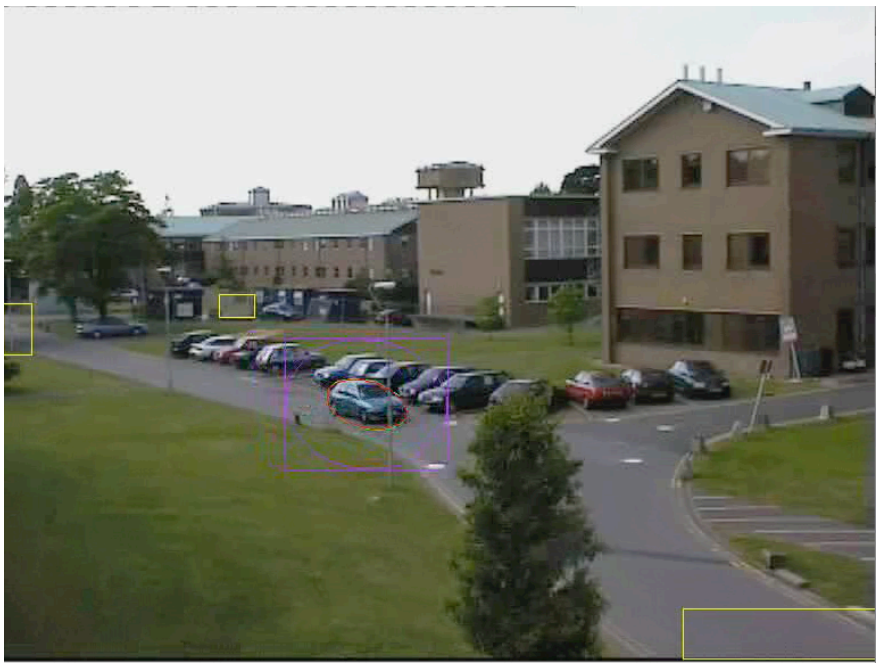


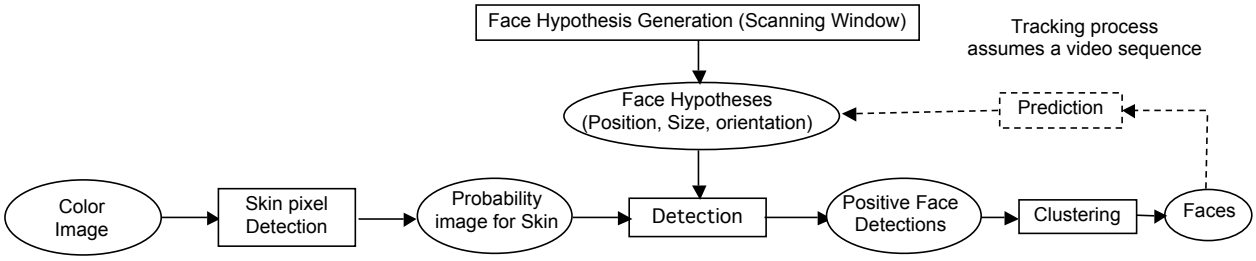
Image from PETS 2000 Data Set. Yellow boxes are Detection Regions.

Detecting Blobs with Color Histograms <show Fame Demo>

3 Face Detection using Skin Color

Skin color can be used to construct a simple detector for skin pixels in images. Color skin pixels can then be used to detect and track "blobs" that represent faces, hands and other skin colored regions in images.

The detector works by first computing the probability that each pixel contains skin. A sliding window (Region of Interest or ROI) is then scanned over the image. At each position, a weighted sum of probabilities is determined. Regions for which the weighted sum is above threshold are detected as faces. Adjacent face detections are grouped to form a single face detection.



Algorithm:
1) Compute probability of skin at each pixel.
2) Test for faces at possible positions and sizes (scanning Window).
3) Cluster adjacent detections.
4) Estimate precise face position, size and orientation from clusters of detections.

The function typically returns a "likelihood" estimate for a face at many adjacent positions, orientations and sizes.

We can break the problem down into 3 challenges.
Each challenge requires experimentally determining a set of parameters.

3.1 Challenge 1: Detecting skin pixels with color

Base Line: Ratio of RGB histograms
The first challenge is to transform a color (RGB) image, *P(i,j)* into an image where each pixel provides an estimate of the probability of skin, *S(i,j)*.

Assume a color image : $P(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}(i,j)$

The algorithm will use a lookup table to convert color to probability.

$$S(i,j) \leftarrow L(\vec{P}(i,j))$$

The lookup table is constructed as a ratio of histograms, as explained below.
We can improve the results can be obtained by using a normalized color space as well as by tuning the quantization of the histogram to the data.

In the following, assume that we have a color image, where each pixel *(i,j)* is a color vector, $P(i,j)$, composed of 3 integers between 0 and 255 representing Red, Green and Blue.

Suppose that we have N color images of size CxR pixels, $P_n(i,j)$.
This gives a total of $M = C \times R \times N$ pixels.
Call this set of pixels $\vec{X}_m \leftarrow S_n(i,j))$ where $m = i \cdot j \cdot n$

Suppose that we have a ground truth function $y(X_m)$ that tells us whether each pixel is target (P or Positive) or not target (N or Negative). A subset of M_T pixels that belong to a target class, *T*.

We allocate two tables $h(\vec{X})$ and $h_T(\vec{X})$ and use these to construct two histograms.
we can also write this as $\forall_m h(\vec{X}_m) = h(\vec{X}_m) + 1$

and $\forall_m y(\vec{X}_m) = P : h_T(\vec{X}_m) = h_T(\vec{X}_m) + 1 \; ; \; M_T \leftarrow M_T + 1$

For a color vector, $\vec{X} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$ we have two probabilities:

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

Bayes rule tells us that we can estimate the probability that a pixel belongs to target class (Skin) given its color, $\vec{X}$ as:

$$P(Skin \mid \vec{X}) = \frac{P(\vec{X} \mid \text{Skin}) P(\text{Skin})}{P(\vec{X})}$$

P(Skin) is the probability that a pixel belongs to the target class.
This can be estimated from the training data by:

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

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

$$P(Skin \mid \vec{X}) = \frac{P(\vec{X} \mid Skin)P(Skin)}{P(\vec{X})} = \frac{\frac{1}{M_T} h_t(\vec{X}) \cdot \frac{M_t}{M}}{\frac{1}{M} h(\vec{X})} = \frac{h_T(\vec{X})}{h(\vec{X})}$$

We can use this to compute a lookup table $L_{Skin}(\vec{X}) = \frac{h_T(\vec{X})}{h(\vec{X})}$

if $h(\vec{X}) = 0$ then $h_{Skin}(\vec{X}) = 0$ because $h_{Skin}(\vec{X})$ is a subset of $h(\vec{X})$.
we will need to test this case to avoid an error from divide by 0.

If we ASSUME that a new image, *P(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 *S(i,j)* that the pixel *(i,j)* belongs to class skin.

$$S(i,j) = L_{Skin}(P(i,j))$$

Details:

1) Alternative color codings may provide better recognition.
2) Different color quantizations may provide better recognition.

In this example, $h(\vec{X})$ and $h_T(\vec{X})$ are composed of $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ cells.
We define this as the Capacity of the histogram, V

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

In general, $V = Q^D$ where Q is the number of values per feature and D is the number of features.

This can result in a very sparse histogram. The reliability can be improved by using more training images from more cases.

A naive statistics view says to have at least 10 training samples for histogram cell.
That is $M \geq 10\ V$. However, it is often easier to work with powers of 2.
For example, $2^3 = 8 \approx 10$ which is approximately 10.
This suggest that we required $M \geq 8\ V = 2^3\ V$.

Thus we would need $2^3 \cdot 2^{24} = 2^{27}$ training pixels. 2^7 Meg.
(Note that a 1024 x 1024 image contains 2^{20} pixels. This is the definition of 1 Meg)

Obtaining 2^7 Meg pixels is not a problem for $P(\vec{X}) = \frac{1}{M} h(\vec{X})$ but may be a problem for training the histogram of target pixels $P(\vec{X} \mid Skin) = \frac{1}{M_T} h_T(\vec{X})$.

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? Two approaches:
We can reduce the number of values, Q, for each feature, or
we can reduce the number of features.

For example, for many color images, Q=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 physics to look for features that are "invariant".

Variation: Detection with Chrominance

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

Several methods exist 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 computed by dividing R and G by luminance. These are often called "r" and "g" in the literature.

Luminance: L= R+G+B

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

The terms r and g have values between 0 and 1. To count statistics we need integer values. For this, it is common to represent r and g as integer numbers coded with Q values between 0 and Q–1 by :

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

From experience, Q = 32 color values seems to work well for skin with most web cameras, but this may change for certain data sets and should be experimentally verified.

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

Suppose we have a set of N training images $\{X_n(i,j)\}$ of size *RxC* where each pixel is an RGB color vector. This gives a total of $M=NxIxJ$ color pixels.

Suppose that M_T of these are labeled as skin pixels i.e. $y(X_n(i,j)) = P$
We allocate two tables: $h(r,g)$ and $h_T(r,g)$ of size Q xQ.
As before:

For all *i,j,n* in the training set $\{X_n(i,j)\}$:
BEGIN

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

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

IF $(y(X_n(i,j) == P)$ *THEN* $h_T(r,g) = h_T(r,g)+1$; $M_T \leftarrow M_T + 1$

END

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

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

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

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

Where *P(i,j)* is an image in which each pixel is a probability value from 0 to 1.
Probabilities can be expressed, of course be expressed as integers by multiplying by a quantization value (Q).

Free Parameters

A number of parameters remain unspecified. Specification of these parameters generally depends on the application domain and the training data. Often these parameters make it possible to trade off error rates for computing time. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

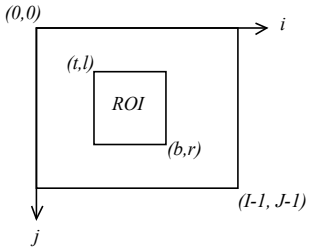
Free parameters for color based skin detection include:

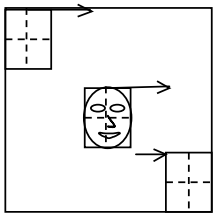
- The use of color coding (eg, RGB, rg, other codings for chrominance)
- Value of Q – the quantification for color color values.
- Training Data – different training and test regimes with variations such as the number of folds, and whether to train with a subset of the folds or all folds.

3.2 Challenge 2: Detecting Faces with Skin Color

We can detect faces from the mass of skin detections within 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. This is not possible for our example, because our training and test data are restricted to static images.

In many vision detectors the ROI is simply a sliding window that scans the entire image as explained above. This is the technique used, for example, with the Viola Jones Face Detector.





As before, the ROI is a rectangle represented by four coordinates: (left, top, right, bottom) or (*l, t, r, b*). Alternatively we can represent the ROI as the center, width and height of the ROI: c_i, c_j, *w, h.*

For detection in static images, we will need to test a range of ROIs with different positions and sizes. The scanning step size (s) may also be varied. This may be the same size for row and column direction, or different step sizes may be used (say s_i and s_j)

Baseline: Sliding Window Detector

We can compute the ROI parameters for a face as a bounding box for an ellipse.

Let us define a face hypothesis as $\vec{X} = \begin{pmatrix} c_i \\ c_j \\ w \\ h \end{pmatrix}$

Since faces are generally vertical, we may assume that the ellipse is oriented with the major axis aligned with the column direction (j). The bounding box ROI is:

$$t = c_j - \frac{h-1}{2}, \qquad b = c_j + \frac{h-1}{2}, \qquad l = c_i - \frac{w-1}{2}, \qquad r = c_i + \frac{w-1}{2}$$

The likelihood of a face at a position (c_i, c_j) of size (w,h) is:

$$g(\vec{X}) = \frac{1}{w \cdot h} \sum_{i=l}^{r} \sum_{j=t}^{b} P(i,j)$$

We can bias this likelihood to make a decision:

IF $g(\vec{X}_m) + B > 0.5$ THEN $R(\vec{X}_m) = P$ else $R(\vec{X}_m) = N$

And of course

IF $R(\vec{X}_m) = y(\vec{X}_m)$ THEN T else F.

This technique will detect "faces" for a range of positions and sizes. As long at the detections overlap with the face ellipse in the data-base they are TRUE detections.

The problem with this technique is that faces are not square. The ROI gives equal weight to corners as the center. We can do better by weighting the pixels using a Gaussian function. This is called a "robust estimator" because it tends to reject outliers.

Variation: Detection using a Gaussian mask.

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. A commonly practice is to use a Gaussian Window to suppress signals outside of the fixated region. A similar technique is used to suppress outliers for robust estimation.

A Gaussian window has the form:

$$G(\vec{P}; \vec{\mu}, \Sigma) = e^{-\frac{1}{2}\left(\vec{P}-\vec{\mu}\right)^T \Sigma^{-1} \left(\vec{P}-\vec{\mu}\right)}$$

Where $\vec{P} = \begin{pmatrix} i \\ j \end{pmatrix}$ represents image positions, and $\vec{\mu} = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ is the center position of the window.

The covariance matrix of the window is: $\Sigma = \begin{pmatrix} \sigma_i^2 & \sigma_{ij} \\ \sigma_{ij} & \sigma_j^2 \end{pmatrix}$

The coefficients have a value of 1 at the center of the mask, and taper to 0.1 at a distance of 2σ and to 0.01 at a distance of 3σ.

We can use the parameters of the window to define a face hypothesis. $\vec{X} = \begin{pmatrix} \mu_i \\ \mu_j \\ \sigma_i^2 \\ \sigma_j^2 \\ \sigma_{ij} \end{pmatrix}$

Normally the Gaussian mask should be computed within a ROI determined by a bounding box. Typically the bounding box should be at least 2σ (standard deviations), but if speed is more important the false negatives, then computation time can be reduced by using a smaller ROI, down to as small as 1 standard deviation.

We can define the ROI as a 2σ bounding box:

$$t = c_j - 2\sigma_j, \qquad b = c_j + 2\sigma_j, \qquad l = c_i - 2\sigma_i, \qquad r = c_i + 2\sigma_i$$

To evaluate a face hypothesis, we compute the face likelihood as a weighed sum of the skin probabilities by the Gaussian mask.

$$g(\vec{X}) = \sum_{i=l}^{r} \sum_{j=t}^{b} P(i,j) G(i,j;\vec{X})$$

As before, the discriminant, $g(\vec{X})$, has a value between 0 and 1. This is the likelihood that a face may be found at $\vec{X}$.

As before, faces are detected as:

IF $g(\vec{X}_m) + B > 0.5$ THEN $R(\vec{X}_m) = P$ else $R(\vec{X}_m) = N$

Free parameters to test

As with color skin detection a number of parameters remain unspecified. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

Free parameters for color face detection include:
For a simple scanning window, these include:

- Width and height of ROI
- Range of positions
- Step size for scanning windows
- The percentage of overlap with the ground truth that is considered a TRUE detection.

For a Gaussian detection window, parameters also include

- The width and height of the ROI compared to the standard deviations of the principal axis.
- The range and step sizes for orientation of the Gaussian window.

3.3 Challenge 3: Face Localization

Detection vs Localization

Detection: A face is considered to be "detected" if a positive detection is found at a position that overlaps the face in the ground truth. The simplest form of test is to verify that the location of the face is within the ellipse of the ground truth label for a face in the image.

For a face hypothesis at $\vec{X} = \begin{pmatrix} c_i \\ c_j \\ w \\ h \end{pmatrix}$ a face is detected if $R(\vec{X})$=P

The detection is TRUE if $R(\vec{X})=y(\vec{X})$

Using the ground truth $y(\vec{X}) = \begin{cases} P & if(\frac{(x-c_x)^2}{r_a^2} + \frac{(y-c_y)^2}{r_b^2} \le 1 \\ N & otherwise \end{cases}$

The Hypothesis is a TRUE POSITIVE detection if $y(\vec{X})$=P and $R(\vec{X})=y(\vec{X})$.

A large number of TP faces will be detected for each ground truth face. If the search space includes multiple scales and orientations, than the number of detected faces will be even larger. All of these correspond to the same face!

Localization (more precisely parameter estimation) specifies precisely where, and with what parameters, the face may be found. There should be only ONE face located for each true face. The face should be located at a position, size and orientation as close to the true face as possible. When searching at multiple scales and orientations, each detection has the form of an error vector.

A distance metric, relating degrees and scale change to position is required to reduce this to a single number. For discrete samples the distance metric can provided implicitly by the sample step size in scale, orientation and position.

We can obtain a location (or parameter estimation) from multiple positive detections by suppressing detections for which the discriminant, $g(\vec{X})$ is not a local maximum. This requires specifying a measure for locality.

We can also estimate the parameters of the face from the moments of a "cloud" of detections at multiple positions, scales and orientations. This is the same robust

estimation technique that we used above, extended to robustly estimate position, size and orientation.

Baseline: Localization by non-maximum suppression

In order to suppress non-maximal detections, the easiest method is to build a list of detections hypotheses, $\{\vec{X}\}$ over the desired range of positions, scales, orientations and any other parameters, and then filter this list to remove any hypothesis for which the discriminant not a local maximum. (A maximum within a some distance R.)

$\forall \vec{X}_i, \vec{X}_j \in \{\vec{X}\}$: IF $Dist(\vec{X}_i, \vec{X}_j) < R$ AND $g(\vec{X}_i) < g(\vec{X}_j)$ THEN $\{\vec{X}\} \leftarrow \{\vec{X}\} - \vec{X}_i$

This requires defining some notion of distance that includes position, scale and orientation. This is generally done with regard to an expected range of positions, orientations and scales over which a single face would be detected. For example, using a Mahalanobis distance (Distance normalized by covariance).

Variation: Localization by Robust Estimation

We can use robust estimation to estimate the most likely parameters from a set of detections. To do this, we will calculate the weighted moments from the set of detections.

Suppose that we have a set of N detections: $\{\vec{X}_n\}$ and for each detection we have a discriminant $g(\vec{X}_n)$.

The mass of the detections is $M = \sum_{n=1}^{N} g(\vec{X}_n)$. This is the zeroth moment of the set of detections.

The "expected value" for $\vec{X}_n$ is $E\{\vec{X}\} = \frac{1}{M} \sum_{n=1}^{N} g(\vec{X}_n) \cdot \vec{X}_n$

This is the first moment (or center of gravity) of the values of $\{\vec{X}_n\}$.

The expected value is a vector of first moments for each parameter:

For D parameters, the center of gravity is a vector

$$\vec{\mu} = E\{\vec{X}\} = \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)\vec{X}_n = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_D \end{pmatrix} = \begin{pmatrix} \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{1n} \\ \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{2n} \\ \dots \\ \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{Dn} \end{pmatrix}$$

The vector $\vec{\mu}_n$ the vector of weighted averages for the components of $\vec{X}_n$.

If there is only one face inside the set $\{\vec{X}_n\}$ of positive detections then $\vec{\mu}$ provides the most likely estimate for set of parameters the detection, (e.g. position, size and orientation).

In the case of multiple faces, the best estimate for each of the faces can be determined using the Expectation Maximization (EM) algorithm. Alternatively, the estimation may be limited to faces detections within a small region of the image.

Free parameters to test

As with color face detection a number of parameters remain unspecified. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

Free parameters for color face detection include:
For a simple scanning window, these include:

- Number of Faces in the image
- Range of positions, size and orientations to test
- Distance to use for non-maximal suppression.
- Size of the region used to cluster faces detections.