Computer Vision

James L. Crowley

M2R MoSIG option GVR

Fall Semester
10 October 2014

Lesson 3

Detecting and Tracking Objects with Color

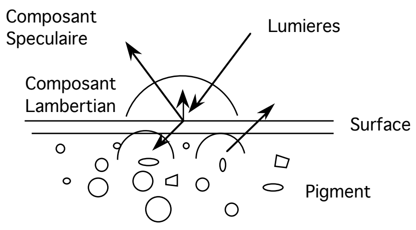
<u>Lesson Outline</u>:

1 Detection and Tracking using Color 2
1.1 Separating Specular and Lambertian Reflection 2
1.2 Histograms as an Estimation of Probability of Color 3
1.3 Bayesian Object Detection with Color 4
1.4 Color Skin Detection 6
2 Bayesian Tracking of Gaussian Blobs 8
2.1 Moment Calculations for Blobs 8
2.2 Bayesian Tracking 11
2.3 Temporal Prediction 11
2.4 Detecting the target 13
2.5 Updating the Estimated Blob Parameters 13
2.6 Managing Lost Targets 14
3 The Kalman Filter 15
3.1 State Vector 16
3.2 Confidence and Uncertainty 16
3.3 State Estimation 19
3.4 Summary of Kalman filter equations. 20

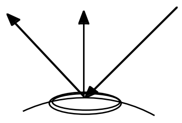
1 Detection and Tracking using Color

Recall the Bi-Chromatic 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 is generally determined by surface orientation, while color is determined by Pigment.



1.1 Separating Specular and Lambertian Reflection.

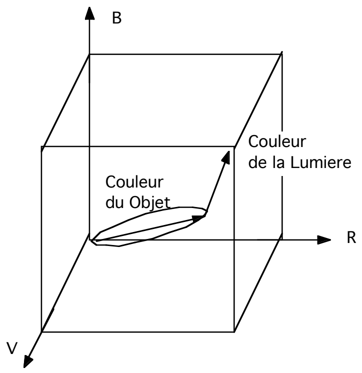
Consider what happens at a specular reflection.

The specularity has the same spectrum as the illumination.

The rest of the object has a spectrum that is the product of illumination and pigments. Suppose that the image be composed of color pixels, $\vec{c}(i,j)$

We can construct an RGB color histogram using a 3 Dimensional table h(R, G, B).

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



Suppose the image contains only a single (non-planar) object that obeys a bi-chromatic reflection model. If the image contains a specular reflection, then two clear axes will emerge.

One axis from the origin to the RGB of the product of the illumination and the source. The other axis towards the RGB representing the illumination.

We can fit lines to these two axes. The dominant line indicates the object pigment color modified by the illumination; $P(\lambda)S(\lambda)$. The line intersects the edge of the cube at this color.

The second (less dominant line) indicates the source color $S(\lambda)$. Source color can be observed from the point where this line exits the cube.

Projecting ALL pixels onto these two lines gives two images:

1) A Lambertian image (without the specularity) and
2) A specular image (without the Lambertian component).

Lesson: color statistics, particularly histograms, can provide powerful analysis tools .

1.2 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 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 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.

1.3 Bayesian Object Detection with Color

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 ;

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

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

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 T(i,j) that the pixel (i,j) belongs to the target T.

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

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

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

1.4 Color Skin Detection

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

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

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.

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 $c_k(i,j)$ is skin THEN
        $h_{skin}(r,g) = h_{skin}(r,g) + 1$

END
```

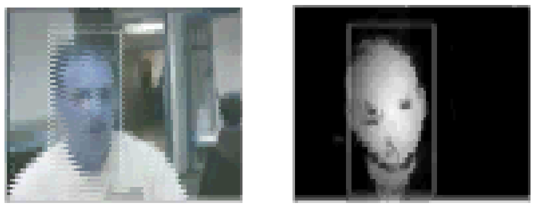
As before, we can obtain a lookup table $L_{skin}(r,g)$ that gives the probability that a pixel is skin.

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

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

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

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


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

We can improve the detection by tracking skin colored 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: *T(i,j),* where for each pixel:

$$T(i,j) = L_T(r(i,j), g(i,j))$$

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. In continuous operation the ROI be provided by tracking.

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

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

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

First moments:

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

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

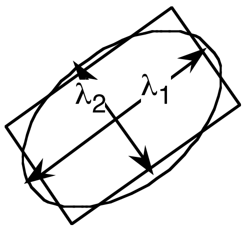
We will use this as the position of the blob.

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 a 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.
The angle θ is a "Covariant" for orientation.

We can use the “eigenvalues”, or characteristic values, λ_1, λ_2, to define the “width and height” of the blob:

For example:

$w=\lambda_1$, $h=\lambda_2$

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

where $x = \mu_i$, $y = \mu_j$, $w=\lambda_1$, $h=\lambda_2$ and $CF = \frac{S}{(b-t)(r-l)}$

The confidence (CF) can be seen as the “Likelihood” that the model for the blob is correct.

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". The vector $\vec{X}$ provides the “model” for the target blob:

Blob model: $\vec{X} = \begin{pmatrix} x \\ y \\ w \\ h \\ \theta \end{pmatrix}$ Precision: $P = \begin{pmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xw}^2 & \sigma_{xh}^2 & \sigma_{x\theta}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yw}^2 & \sigma_{yh}^2 & \sigma_{y\theta}^2 \\ \sigma_{wx}^2 & \sigma_{wy}^2 & \sigma_{ww}^2 & \sigma_{wh}^2 & \sigma_{w\theta}^2 \\ \sigma_{hx}^2 & \sigma_{hy}^2 & \sigma_{hw}^2 & \sigma_{hh}^2 & \sigma_{h\theta}^2 \\ \sigma_{\theta x}^2 & \sigma_{\theta y}^2 & \sigma_{\theta w}^2 & \sigma_{\theta h}^2 & \sigma_{\theta\theta}^2 \end{pmatrix}$

along with CF. (Confidence)

The covariance P expresses our “uncertainty” about the values of each of the parameters. This is also called the “Precision”.

Formally, precision is defined as the 2nd moment of the error of the blob model.
Let $\vec{X}$ be the TRUE (unknown) model and $\hat{\vec{X}}$ be the estimated value. The error, $\vec{E}$, is the difference

$$\vec{E} = \vec{X} - \hat{\vec{X}}$$

and then.

$$P = \vec{E}\vec{E}^T$$

In most cases, the true value is unknown, and so the precision must be estimated from the difference between the predicted and observed blobs.

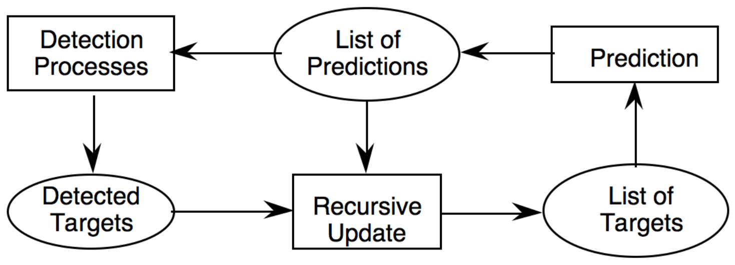
This precision can be initialized to a very small value. Our uncertainty about the precision will grow over time between observations, due to unobserved motion, acceleration, etc. We can estimate this growth by the difference between a predicted blob and its observation. This process will tend to reveal covariance between terms of the matrix.

If possible it is preferred to estimate P from some known values of $\vec{X}$ or from knowledge about the sensors and from "experience".

In the absence of information about the correlation between features in X, the Covariance can be initialised as a diagonal matrix. The tracking process will review any correlations.

2.2 Bayesian Tracking

A Bayesian tracker is a recursive estimator, composed of three phases:
Predict, Detect, Update.



Detection can be provided by detecting the blob using color statistics within a target "Region of Interest" given by a bounding box centered on a previous position. The size of this box is determined by the estimated size of the blob enlarged by the uncertainty *P*.

For this we must predict the new position, detect (observe) the blob, and the update the estimate.

The following is a "zero-th" order Kalman filter. This is the simplest (almost trivial) case of a Bayesian tracker. A first order Kalman filters estimates parameters and their derivatives. The math is more complex but the principles are the same.

2.3 Temporal Prediction

The following describes prediction and estimation (updating).

In the absence of any knowledge of movement, we can predict that the blob will be at the last observed position. This is written as :

$$\vec{X}_t^* \leftarrow \hat{\vec{X}}_{t-1}$$

This is called a "process model". The process model predicts the state vector at time t given the estimate at time t-Δt:

$$X_t^* = \text{argmax}\{ p(X_t^* \mid X_{t-\Delta t})\}$$

This can be written as:

$$X_t^* := \vartheta(\Delta t)\hat{X}_{t-\Delta t} + R$$
$$P_t^* := \vartheta(\Delta t)\hat{P}_{t-\Delta t}\vartheta(\Delta t)^T + Q_x$$

For the simple zeroth order filter $\vartheta(\Delta t)$ is the identity matrix. R is the expected "error" of the process model, due to unmodeled derivatives and noise. The expectation of the error is almost always 0.

$$R = E\{\vec{X} - \hat{\vec{X}}\} = 0$$

Q is the second moment of the process model error.

$$Q = E\{(\vec{X} - \hat{\vec{X}})(\vec{X} - \hat{\vec{X}})^T\}$$

Q captures the loss of precision in the evolution of the process noise.

In a first order Kalman filter this would have been

$$\vec{X}_t^* \leftarrow \vec{X}_{t-1} + \Delta t \vec{X}'_{t-1}$$

where

$$\vec{X}'_{t-1} = \frac{d}{dt}\begin{pmatrix} x \\ y \\ w \\ h \\ \theta \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ w' \\ h' \\ \theta' \end{pmatrix}$$

and Δt is the time step. But let us keep the explanation simple for now.

Because of unobserved velocity and acceleration, etc, the estimate of the blob grows more uncertain with time. This is estimated by a prediction error : $Q_{\Delta t}$

$$C_t^* = \hat{C}_{t-\Delta t} + Q_{\Delta t}\Delta t^2$$

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".

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

2.4 Detecting the target

For each sensor, a predicted sensor signal Y_t^* is generated based on current the estimated system state X_t^*.

$$Y_t^* = argmax\{p(Y_t \mid X_t^*)\}.$$

We obtain this prediction from the predicted target model.

$$Y_t^* = H_X^Y X_t^*$$

In the trivial case of a zeroth order tracker, if $\vec{X}$ and $\vec{Y}$ have the same features than H_X^Y is the identity matrix. Else H_X^Y can "extract" observed features from a more complex model.

In the case of blob detection, we can estimate the bounding box using all 5 parameters of Y_t^*. In this case H_X^Y is the identity matrix.

2.5 Updating the Estimated Blob 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.

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

Where $P_T(i,j)$ is the new observed probability image at the new time, t.
and $(\vec{\mu}_t^*, \Sigma_t^*)$ are extracted from $\vec{X}_t^*$ and C_t^*

$$\vec{\mu}_t^* = \begin{pmatrix} x_t^* \\ y_t^* \end{pmatrix} \text{ and } \Sigma_t^* = \begin{pmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 \end{pmatrix}$$

This is an example of "robust" estimation.

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

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

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 $\hat{C}_t = \begin{pmatrix}\hat{\sigma}_i^2 & \hat{\sigma}_{ij}^2 \\ \hat{\sigma}_{ij}^2 & \hat{\sigma}_j^2\end{pmatrix}$

From which we can update the new estimate : $\hat{\vec{X}}_t$

In the Kalman Filter, this is modeled as :

$X_t = \text{argmax}\{p(X_t \mid X_t^*, Y_t - Y_t^*)\}$

2.6 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 $CF_t \le CF_{min}$ then a target is removed.

The weight α determines the decay rate for lost targets.

3 The Kalman Filter

Three key steps characterise Bayesian estimation problems (including Kalman filters).

1) A process model: The process model predicts the state vector at time t given the estimate at time t-Δt:

$$X_t^* = \text{argmax}\{\, p(X_t^* \mid X_{t-\Delta t})\}$$

2) A sensor model: For each sensor, a predicted sensor signal Y_t^* is generated based on current the estimated system state X_t^*.

$$Y_t^* = \text{argmax}\{p(Y_t \mid X_t^*)\}.$$

3) Re-estimation: A new estimated value, X_t is computed based on information provided by the difference between the predicted and observed sensor values.

$$X_t = \text{argmax}\{p(X_t \mid X_t^*, Y_t - Y_t^*)\}$$

The Kalman filter uses a linear dynamic model to provide these estimates. That is, the process model and sensor models are represented by linear equations. A fixed time step and previously estimated derivative values are used to estimate the current value of the state variables. A quadratic form this same dynamic equation is used to predict the error of the state vector.

A simple zeroth order Kalman filter may be used to track bodies, faces and hands in video sequences. In the following example, let us assume a line segment target whose properties are represented by a "state vector" composed of position, size and orientation (x, y, s, θ) where s is the length (size) of the segment.

A 4x4 covariance matrix is associated with this vector to represent correlations in errors between parameters. Although prediction does not change the estimated position, it does enlarge the uncertainties of the position and size of the expected target. The expected size provides bounds on the sample rate, as we limit the sample rate so that there are at least 8 pixels across an expected target.

3.1 State Vector

The target state vector, $\hat{X}_t$ is composed of the position, scale and orientation of the target. For example, this can represent a human face.

$$\hat{X}_t = \begin{pmatrix} x \\ y \\ s \\ \theta \end{pmatrix}$$

where x, y are the position of the target in pixels
s is the size or scale of the target, and
θ is the image plane orientation of the target.

In the case of a first order filter, each of the parameters is accompanied by first temporal derivative.

$$\hat{X}_t = \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ s \\ \dot{s} \\ \theta \\ \dot{\theta} \end{pmatrix}$$

Where the dot indicates temporal derivative.

$$\dot{x} = \frac{\partial x}{\partial t}$$

The Kalman filter equations are able to use information from the difference of observed and predicted state to estimate the temporal derivatives.

3.2 Confidence and Uncertainty

The state vector is accompanied by a covariance matrix and a confidence factor. The confidence factor is an integer between 0, and a maximum confidence value.

$CFt \in [0, CFmax]$

The position uncertainty (or precision) is the covariance matrix for the state vector

$$P_t = \begin{pmatrix} \sigma^2_{xx} & \sigma^2_{xy} & \sigma^2_{xs} & \sigma^2_{x\theta} \\ \sigma^2_{yx} & \sigma^2_{yy} & \sigma^2_{ys} & \sigma^2_{y\theta} \\ \sigma^2_{sx} & \sigma^2_{sy} & \sigma^2_{ss} & \sigma^2_{s\theta} \\ \sigma^2_{\theta x} & \sigma^2_{\theta y} & \sigma^2_{\theta s} & \sigma^2_{\theta\theta} \end{pmatrix}$$

For the case of a first order filter, this matrix becomes 8 by 8 with covariance between all terms.

For each target, at each time, t, the tracker maintains an estimated state $\hat{X}_t$ as well as its precision $\hat{P}_t$ and confidence factor, CF_t. Based on a previous state and precision $\hat{X}_t$, $\hat{P}_t$ and CF_t as well as the observation $\hat{P}_t$ from detection function accompanied by the observed precision P_y, and detection confidence CF_y.

$$\hat{X}_t, \hat{P}_t, CF_t = F\{X^*_{t+\Delta t}, P^*_{t+\Delta t}, CF_{t-\Delta t}, Y, P_y, CF_y\}$$

Given a target at time t-Δt, the prediction equations predict its new position, and validation gate at time t. The general form of the prediction equations are :

$$\begin{aligned} X^*_t &:= \vartheta(\Delta t)\hat{X}_{t-\Delta t} + R \\ P^*_t &:= \vartheta(\Delta t)\hat{P}_{t-\Delta t}\vartheta(\Delta t)^T + Q_x \end{aligned}$$

These equations are a linear estimation of movement based on a Taylor series approximation.

The term R is a residue that represents higher order (non-estimated) derivatives. This expected value of this term is zero and thus R is commonly omitted. The second moment of R represents the uncertainty due to accelerations (and higher order derivatives). Thus second moments, Q, estimates the loss of precision due to higher order terms.

$$Q = E\ \{R\ R^T\}$$

When included in the prediction, the term Q provides to an additive growth in the validation gate that is translated to the search region. When a target is detected, this growth is disappears in the update phase. However if no target is detected, the result is that the validation gate (and thus the region of interest) grows with each frame until the target is re-acquired or until the target is declared lost.

For a first order filter, the prediction matrix $\vartheta(\Delta t)$ predicts new values as a function of the time step and the estimated derivatives.

$$\vartheta(\Delta t) = \begin{pmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In the case of face tracking, we generally assume that face accelerations are too rapid to be estimated. Thus we may estimate only target position (order 0 tracking) or position and velocity (order 1 tracking). In this case, the prediction matrix, $\varphi(\Delta t)$, becomes a trivial identity matrix:

$$\varphi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In this case our prediction equations are reduced to

$$X_t^* = \hat{X}_{t-\Delta t}$$
$$P_t^* = \hat{P}_{t-\Delta t} + Q_x$$

A validation gate provides a region of interest (ROI), which serves to limit estimation to the region where the target can be detected. For face detection, this region of interest specifies a range of positions and scales and possibly orientations at which the target may be sought. This greatly accelerates processing by avoiding processing unnecessary pixels.

$$ROI = (x_{min}, x_{max}, y_{min}, y_{max}, s_{min}, s_{max}, \theta_{min}, \theta_{max})$$

The ROI is computed using the scale to define the width and height of a rectangular region over which the target will be sought. A ROI based on three standard deviations reasonable size search region. Such a ROI is defined can be defined as

$$x_{min} = x_t - 3\sigma_{xx}$$
$$x_{max} = x_t + 3\sigma_{xx}$$
$$y_{min} = y_t - 3\sigma_{yy}$$
$$y_{max} = y_t + 3\sigma_{yy}$$
$$s_{min} = s_t - 3\sigma_{ss}$$
$$s_{max} = s_t + 3\sigma_{ss}$$
$$s_{min} = s_t - 3\sigma_{ss}$$
$$s_{max} = s_t + 3\sigma_{ss}$$

$$\theta_{min} = \theta_t - 3\sigma_{ss}$$
$$\theta_{max} = \theta_t + 3\sigma_{ss}$$

This can then be used to drive face detection process using a cascade of linear classifiers. The resulting face location is noted as the observed location

$$Y_t = \begin{pmatrix} x \\ y \\ s \\ \theta \end{pmatrix}$$

The difference between the predicted and observed face locations is known as the innovation:

Innovation: $(Y_t - X_t^*)$

For the orientation uncertainty, we propose a fixed value, determined from observing examples of faces, σ_θ. This scale becomes the observed uncertainty: P_y

$$P_y = \begin{pmatrix} \sigma_k^2 & 0 & 0 & 0 \\ 0 & \sigma_k^2 & 0 & 0 \\ 0 & 0 & \sigma_k^2 & 0 \\ 0 & 0 & 0 & \sigma_\theta^2 \end{pmatrix}$$

3.3 State Estimation

This scale is used to update the precision of the face position estimate. The estimated state vector is updated by using the difference between predicted and estimated position as an "innovation". In the case of the 0th order filter, the equations are relatively simple. We first compute a Kalman "Gain matrix:

$$K = P_t^*(P_t^* - P_y)^{-1}$$

from this we can compute the following update equations:

$$\hat{X}_t = X_t^* + K(Y_t - X_t^*)$$
$$\hat{P}_t = P_t^* - KP_t^*$$

The resulting algorithm in the case of a 1st order filter, a transformation matrix, H_x^y used to project X onto the observed variables Y.

3.4 Summary of Kalman filter equations.

1) Process model: Tremporal prediction.

$$X_t^* := \vartheta(\Delta t)\hat{X}_{t-\Delta t} + R$$
$$P_t^* := \vartheta(\Delta t)\hat{P}_{t-\Delta t}\vartheta(\Delta t)^T + Q_x$$

2) Sensor Model: Observation prediction -
H_X^Y predicts observed vector from state vector

$$Y_t^* = H_X^Y X_t^*$$
$$P_y = H_X^Y P_t^* H_X^Y$$

3) Observation and update:

$$K = P_t^*(P_t^* - P_y)^{-1} \text{ (Kalman Gain)}$$
$$\hat{X}_t = X_t^* + K(Y_t - X_t^*)$$
$$\hat{P}_t = P_t^* - KP_t^*$$