Computer Vision

MSc Informatics option GVR
James L. Crowley

Fall Semester 15 November 2012

Lesson 6

Scale Space and Pyramids, Detection and Tracking

Lesson Outline:

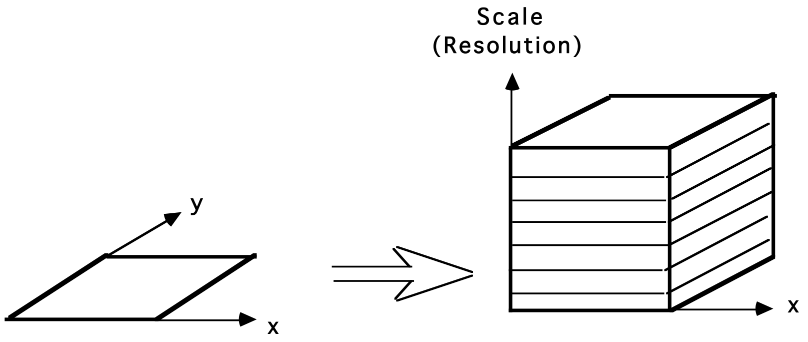
1. Image Scale Space

1.1. Continuous Scale Space.

Let P(x, y) be an image of size W columns by R rows.
Let G(x, y, σ) by a Gaussian function of scale σ^s

Image Scale space is a 3D continuous space P(x, y, s)

$$P(x, y, s) = P * G(x, y, 2^s)$$



Scale space:
Separates global structure from fine detail.
Provides context for recognition.
Can provides local descriptions (features) of the image that are invariant to position, orientation and scale.

Note that the scale axis (s) in scale space is logarithmic

$$s = \mathrm{Log}_2(\sigma) = \mathrm{Log}_2(2^s)$$

A logarithmic scale axis is necessary for scale equivariance.
The appearance of a pattern in the image results in a unique structure in P(x, y, s).
If a shape in an image is made larger by $D = 2^d$

$$p(x,y) \to p(x2^d, y2^d)$$

Then the scale space projection of appearance is shifted by s in scale.

$$P(x,y,s+d) = p(x2^d, y2^d) * G(x2^{s+d}, y2^{s+d}, 2^{s+d})$$

This structure is "equivariant" in position, scale and rotation meaning that the structure has the same form, only shifted in the scale axe.

Translate the pattern by Δx, Δy and the structure translates by Δx, Δy in P(x, y, s).
Rotate by θ in x, y and the structure rotates by θ in P(x, y, s).
Scale by a factor of 2^s, and the structure translates by Δs in P(x, y, s).

1.2. Discrete Scale Space - Scale invariant impulse response.

In a computer, we need to discretize (sample) the axes x, y, and s.

Let P(x, y) is an image array of size WxH pixels, where (x, y) are integers,

We propose to sample scale with a step size of $\Delta\sigma = 2^{1/2}$ so that $\sigma_k=2^{k/2}$
For k=0, to K.

$\sigma_0=1$ is the smallest scale that we can represent.

At k=0 $\sigma_0=2^{0/2}=1$.

K is the largest scale possible: $K=2\text{Log}_2(\min(W, H))$
For larger K the scale parameter σ is larger than the image.

1.3. Spatial Resampling

Because each level of the Gaussian Space has been smoothed by convolution with a Gaussian low pass filter, it is possible to resample the image with a step size that grows with the scale of the Gaussian.

For example, we can use

$$\Delta x_k = 2^{(k-1)/2}$$

With only minimal alliassing.

The result an identical impulse response at each level.
This property is called “scale invariance”.

Diagonal, Square root of two Sampling

With $\sigma_k=2^{k/2}$ the impulse response doubles ever two levels. What happens on the even levels?

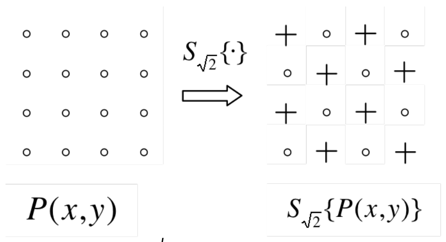
$$\Delta x = 2^{(k-1)/2} = (\sqrt{2})^{k-1}$$

for k odd, $\Delta x_k = \{1, 2, 4, 8\ldots\}$
for k even, $\Delta x_k = \{\sqrt{2}, 2\sqrt{2}, 4\sqrt{2}, 8\sqrt{2}, \ldots\}$

Problem : How can we sample an image at $\Delta x=\sqrt{2}$?

Solution: Sample along diagonals!

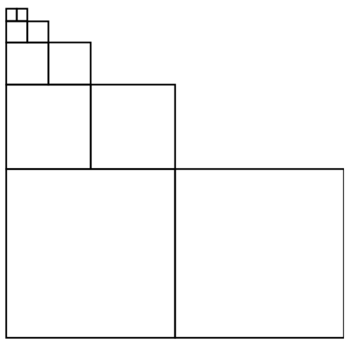
How ? with the diagonal sampling operator $S_{\sqrt{2}}\{\}$

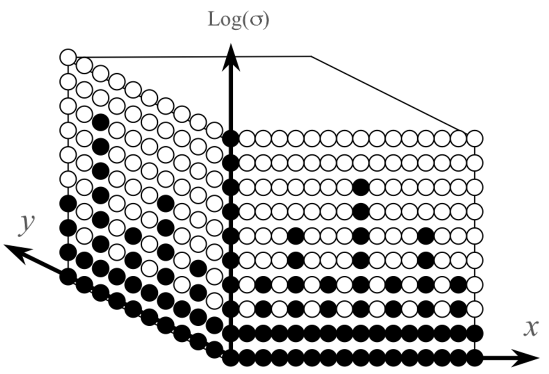


For k even, the √2 resampling operator, $S_{\sqrt{2}}{}^{k}\{\}$, selects even columns of even rows and odd columns of odd rows.

For k odd, diagonal sample operator eliminates every second column (starting with even columns on even rows and odd columns on odd rows). For k odd, resampling eliminates every second row (odd rows).

$$S_{\sqrt{2}^k}\{P\{x,y)\} = \begin{cases} P(x,y) & \text{if } (x+y)^2 \text{ Mod } 2^{k-1} = 0 \\ 0 & \text{otherwise} \end{cases}$$





With root 2 sampling, the number of samples is reduced by half every level. (Normally with S=2 sampling, it would be reduced by 4).

This is illustrated with the following table.

k	$s_k=2^{k/2}$	$\Delta x_k=2^{(k-1)/2}$	Columns	Rows	Samples
0	1	1	W	W	N
1	√2	1	W	W	N
2	2	√2	W/2	W	N/2
3	2√2	2	W/2	W/2	N/4
4	4	2√2	W/4	W/2	N/8
5	4√2	4	W/4	W/4	N/16
6	8	4√2	W/8	W/4	N/32
7	8√2	8	W/8	W/8	N/64
8	16	8√2	W/16	W/8	N/128

For an image of size N=WxH, number of samples, if we disregard level k=0, then the pyramid has P samples.

$$P = N \times (1 + ½ + ¼ + \ldots) = 2N$$

But note that the for the last few levels, $\sigma_k >$ Min(W, H), and the level is dominated by boundary effects and not usable.

This is true when Min(W, H) < $2^{(k-1)/2}$

1.4. Using the Pyramid to compute image derivatives

Last week we saw thatthe derivatives can be computed by convolving the image with derivatives of Gaussians

$$P_x(x,y) \approx P * G_x(x,y,\sigma)$$

With the Pyramid, derivatives can be obtained directly by sum and difference of the resampled pixels.

Let $i = x/\Delta x_k$ and $j = y/\Delta y_k$

Then

$$P_x(i,j,k) \approx P(i+1,j,k) - P(i-1,j,k)$$
$$P_y(i,j,k) \approx P(i,j+1,k) - P(i,j-1,k)$$
$$P_{xx}(i,j,k) \approx P(i+1,j,k) - 2P(i,j,k) + P(i-1,j,k)$$
$$P_{yy}(i,j,k) \approx P(i,j+1,k) - 2P(i,j,k) + P(i,j-1,k)$$
$$P_{xy}(i,j,k) \approx P(i+1,j+1,k) - P(i-1,j+1,k) - P(i+1,j-1,k) + P(i-1,j-1,k)$$

These are sometimes referred to as "Receptive Fields"

Laplacien: $\nabla^2 P(x,y,k) = P * \nabla^2 G(x,y,\sigma_k) = P_{xx}(x,y,k) + P_{yy}(x,y,k)$

For a pyramid we can use the diffusion Equation to show:

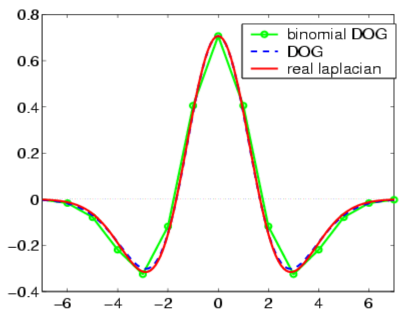
$$\nabla^2 G_x(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial \sigma}$$

As a consequence: $\nabla^2 G(x, y, \sigma) \approx G(x, y, \sigma_1) - G(x, y, \sigma_2)$

This typically requires $\sigma_1 \geq \sqrt{2}\ \sigma_2$

Thus it is common to use:

$$\nabla^2 P(x, y, k) \approx P(x, y, k) - P(x, y, k-1)$$

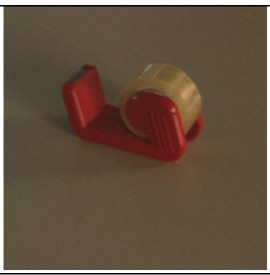
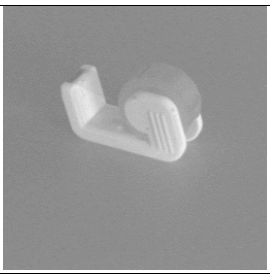
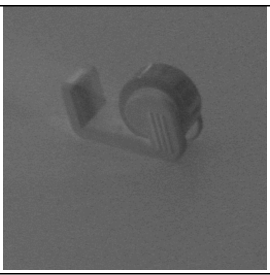


1.5. Color Opponent Scale Space

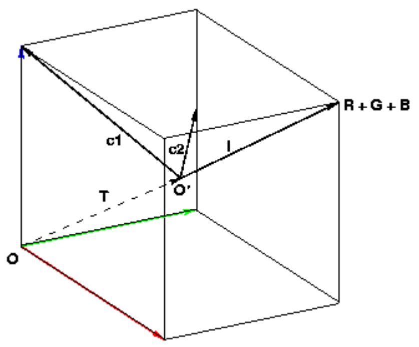
In lesson 3 we saw that a color opponent space was useful for illumination invariance

$$(R, G, B) \Rightarrow (L, C_1, C_2) \qquad \begin{pmatrix} L \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 0.33 & 0.33 & 0.33 \\ -0.5 & -0.5 & 1 \\ 0.5 & -0.5 & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

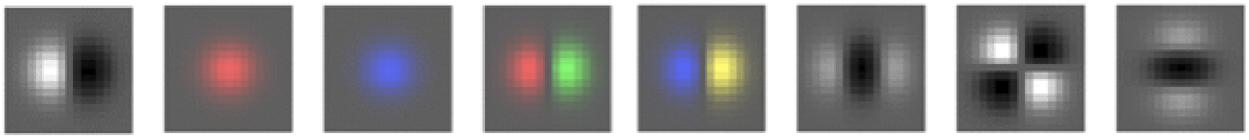
This representation separates luminance and chrominance.

				
RGB		B-W	R-G	R+G-B

Color opponent space can be used to build receptive fields that can be steered in color



$$\begin{pmatrix} L \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 0.33 & 0.33 & 0.33 \\ -0.5 & -0.5 & 1 \\ 0.5 & -0.5 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1 R \\ \alpha_2 G \\ \alpha_3 B \end{pmatrix}$$



We then compute 3 pyramids : L(x, y, k), C_1(x, y, k), and C_2(x, y, k),

This gives us a feature vector for appearance:

$$\vec{A}(x,y,k)=\begin{bmatrix} G_x^{L\sigma_k} \\ G^{C_1\sigma_k} \\ G^{C_2\sigma_k} \\ G_x^{C_1\sigma_k} \\ G_x^{C_2\sigma_k} \\ G_{xx}^{L\sigma_k} \\ G_{xy}^{L\sigma_k} \\ G_{yy}^{L\sigma_k} \end{bmatrix}$$

This can be generalized to include multiple scales.

2. Pixel Level Detection using Color and Appearance

Recall from Lecture 3 that we used Baye's Rule and histograms to determine the probability that a pixel was skin. Probabilities can be grouped into "blobs" and tracked with a Bayesian Tracker.

Reminder:
Skin pigment is generally always the same color. Skin chrominance is invariant with illumination intensity.

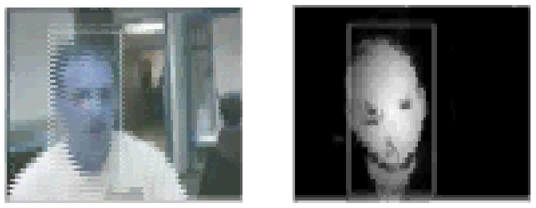
Chrominance : $c_1 = r = \frac{R}{R+G+B} \qquad c_2 = g = \frac{G}{R+G+B}$

We can map this to N+1 values between 0 and N

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

From Bayes rule:

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



We can use this to convert each color pixel *c(i,j)* to a probability, *p(i,j)*, by table lookup.

This approach can be generalized for other local features.
For example, the vector of Gradient Derivatives.

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

2.1. Histograms of Receptive Fields

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

$$\vec{V}(i,j;\sigma_i,\theta_i) = P(i,j) * (G_x, G_{xx}, G_{xy}, G_{yy}) \text{ at } \sigma_i \text{ and } \theta_i.$$

ATTENTION. The histogram must have sufficient samples M.

$$M \geq 10\ Q \geq 10\ N^{D.}$$

For the above example: D = 4.

Here is a table of numbers of cells in a histogram of D dimensions of N values.

N \ d	1	2	3	4	5	6
2	2^{1}	2^{2}	2^{3}	2^{4}	2^{5}	2^{6}
4	2^{2}	2^{4}	2^{6}	2^{8}	2^{10} =1 Kilo	2^{12} =2 Kilo
8	2^{3}	2^{6}	2^{9}	2^{12}	2^{15}	2^{18}
16	2^{4}	2^{8}	2^{12}	2^{16}	2^{20} = 1 Meg	2^{24} = 4 Meg
32	2^{5}	2^{10} =1 Kilo	2^{15}	2^{20} = 1 Meg	2^{25}	2^{30} = 1 Gig
64	2^{6}	2^{12}	2^{18}	2^{24}	2^{30} = 1 Gig	236
128	2^{7}	2^{14}	2^{21} = 2 Meg	2^{28}	2^{35}	2^{42} =2 Tera
256	2^{8}	2^{16}	2^{24}	2^{32} = 2 Gig	2^{40} = 1 Tera	248

Consider the chromatic receptive fields normalized in scale and orientation σ_i and θ_i.

$$\vec{P}_{\sigma,\theta} = (P_x^{L},\ P_x^{C_1},\ P_x^{C_2},\ P_{xx}^{L},\ P_{xy}^{L},\ P_{xx}^{C_1},\ P_{xx}^{C_2})$$

D= 7.

$$p(\text{objet}(i,j)\ |\ \vec{V}(i,j)\ ) = \frac{p(\vec{V}(i,j) \mid object(i,j)}{p(object(i,j))} p(\vec{V}(i,j) \approx \frac{h_o(\vec{V}(i,j))}{h(\vec{V}(i,j))}$$

3. Gaussian Blob Tracking

To construct a Bayesian tracker, we need to represent clouds of pixels with high probability of being a target. To do this we represent such clouds as "Gaussian Blobs".

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

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

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

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

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

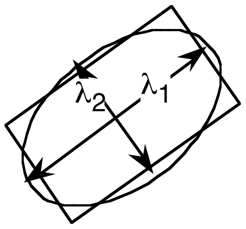
Position is the center of gravity: (μ_i, μ_j)

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

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

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

The principle components (λ_1, λ_2) determine the length and width.
The principle direction determines the orientation of the length.
We can discover these by principle components analysis.



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

where

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

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

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

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

and

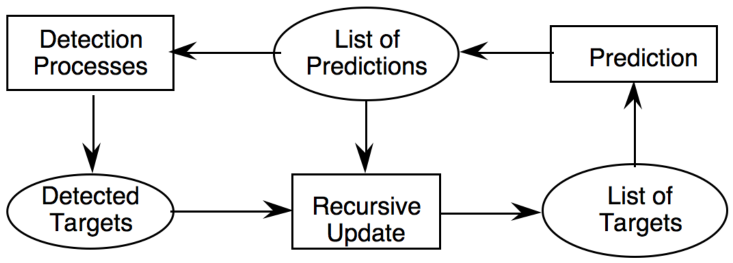
$$CF = \frac{S}{(b-t)(r-l)}$$

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

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

3.2. Bayesian Estimation

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



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

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

Our Gaussian blob is

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

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

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

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

Detected target pixels:

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

We then estimate the new position and covariance as before:

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

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

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

3.3. Temporal Prediction

The scene evolves. Targets move.

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

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

If we have a temporal model, we can estimate

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

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

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

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

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

3.4. Managing Lost Targets

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

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

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

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

If $CF_t \leq 0$ then a target is removed.

4. 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 this model, targets properties are represented by a "state vector" composed of position, scale and orientation (x, y, σ, θ). 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.

4.1. State Vector

The target state vector, $\hat{X}_t$ is composed of the position, scale and orientation of the target.

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

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.

4.2. 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

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

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 :

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

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 face 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.

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

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: $(Y_t - X_t^*)$

The resulting face location is accompanied by an estimate of its precision, essentially scale of the scale at which the face was detected, σ_k. For scale uncertainty, we propose a value of $\sigma_s = \sqrt{2}$. This corresponds to one level of a binomial pyramid. Alternate values can be used depending on the observed uncertainty in size. 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

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

4.3. 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.