MAEG 5720:
Computer Vision in Practice

Lecture 6:

Tracking

Dr. Terry Chang

2021-2022

Semester 1

香港中文大學
The Chinese University of Hong Kong

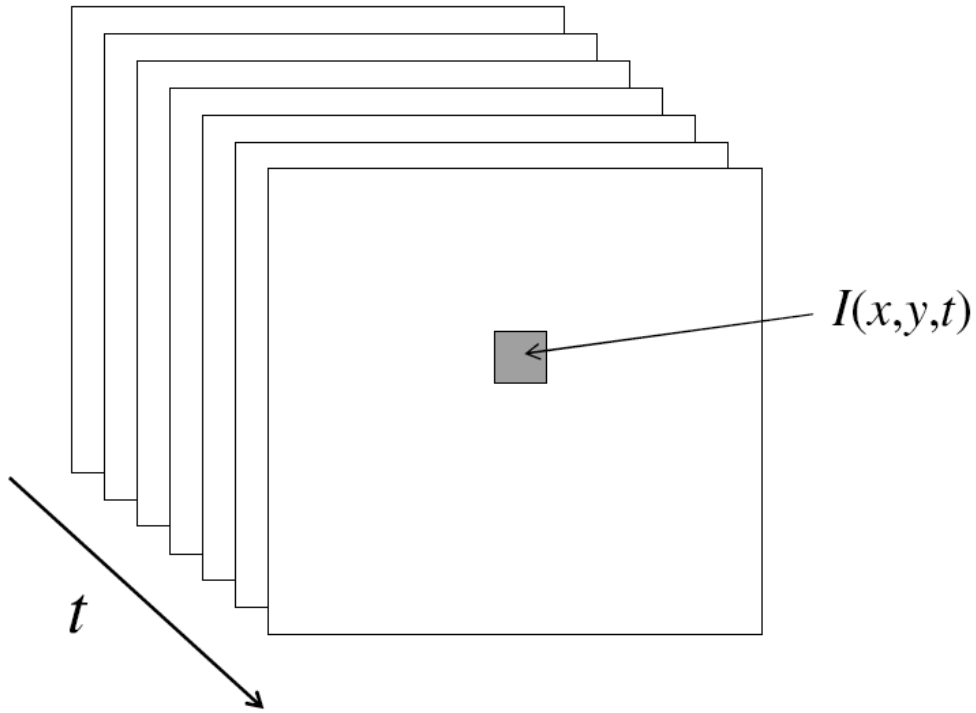
Department of Mechanical and Automation Engineering
機械與自動化工程學系

Last Lecture

- From Image to Sequence
- Definition of Optical Flow
- Horn and Schunck Flow
- Lucas Kanade Flow
- Course-to-fine approach

Recall - From Images to Videos

- A Video is a sequence of frames captured over time.
- Now the image data becomes a function of space(x, y) and time (t)



Recall - Horn & Schunk (Regularization)

- Brightness Constancy Equation

$$I_x u + I_y v + I_t = 0$$

- How to solve the equation?

- Imposing smoothness Constraints

$$\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2$$

- The gradient of optical flow velocity u and v should small

Recall - Horn and Schunck Algorithm

begin

for $j := 1$ **to** N **do** **for** $i := 1$ **to** M **do** **begin**

calculate the values $E_x(i,j,t)$, $E_y(i,j,t)$, and $E_t(i,j,t)$ using a selected approximation formula;

{ special cases for image points at the image border have to be taken into account }

initialize the values $u(i,j)$ and $v(i,j)$ with zero

end {for};

choose a suitable weighting value λ; { e.g. $\lambda = 10$ }

choose a suitable number $n_0 \geq 1$ of iterations; { $n_0 = 8$ }

$n := 1$; { iteration counter }

while $n \leq n_0$ **do begin**

for $j := 1$ **to** N **do** **for** $i := 1$ **to** M **do** **begin**

$$\bar{u} := \frac{1}{4}\left(u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1)\right);$$

$$\bar{v} := \frac{1}{4}\left(v(i-1,j) + v(i+1,j) + v(i,j-1) + v(i,j+1)\right);$$

{ treat image points at the image border separately }

$$\alpha := \frac{E_x(i,j,t)\bar{u} + E_y(i,j,t)\bar{v} + E_t(i,j,t)}{1 + \lambda\left(E_x^2(i,j,t) + E_y^2(i,j,t)\right)} \cdot \lambda \;;$$

$$u(i,j) := \bar{u} - \alpha \cdot E_x(i,j,t) \;; \quad v(i,j) := \bar{v} - \alpha \cdot E_y(i,j,t)$$

end {for};

$n := n + 1$

end {while}

end;

Horn & Schunck - Matlab Example

- Compute Derivatives

```
fx = conv2(im1,[-1 1; -1 1], 'valid'); % partial on x
fy = conv2(im1, [-1 -1; 1 1], 'valid'); % partial on y
ft = conv2(im1, ones(2), 'valid') + conv2(im2, -ones(2), 'valid'); % partial on t
```

- Laplacian Mask and averaging

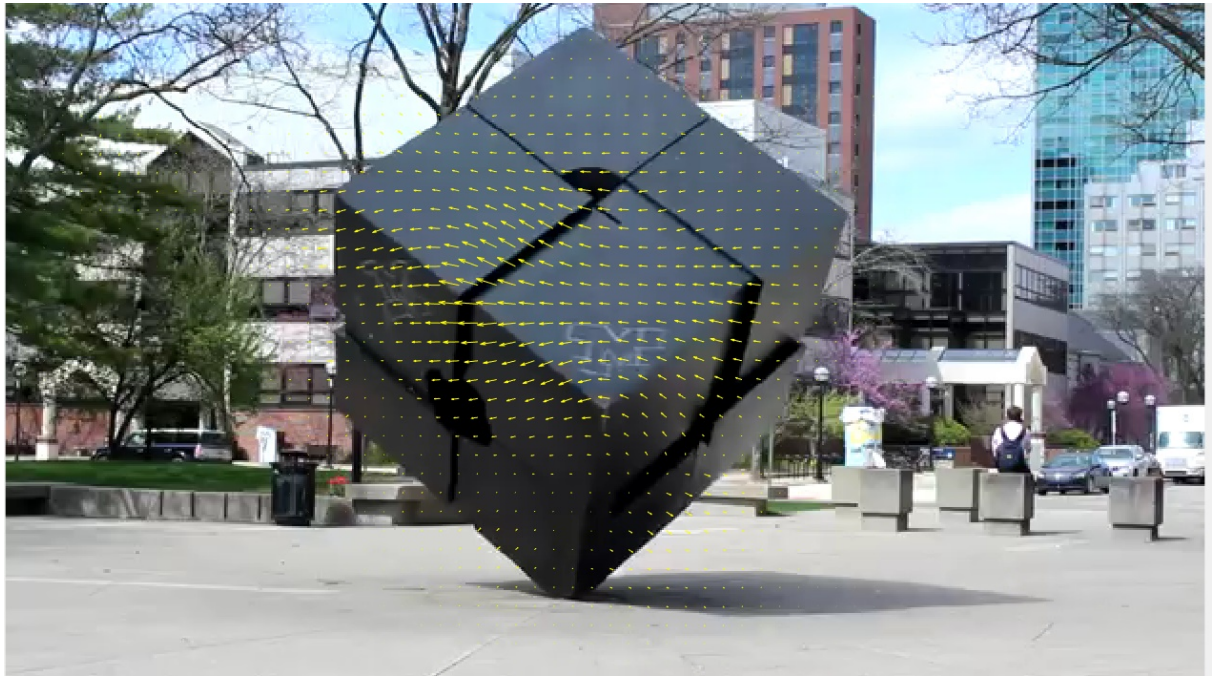
```
kernel_1=[0 1/4 0;1/4 0 1/4;0 1/4 0];

% or

kernel_1=[1/12 1/6 1/12;1/6 0 1/6;1/12 1/6 1/12];
```

```
uAvg=conv2(u,kernel_1,'same');
vAvg=conv2(v,kernel_1,'same');
```

- Iteration

```
alpha = (fx.*uAvg+fy.*vAvg+ft)./(1+lamda*(fx.^2+fy.^2));
u = uAvg-alpha.*fx;
v = vAvg-alpha.*fy;
```

Result

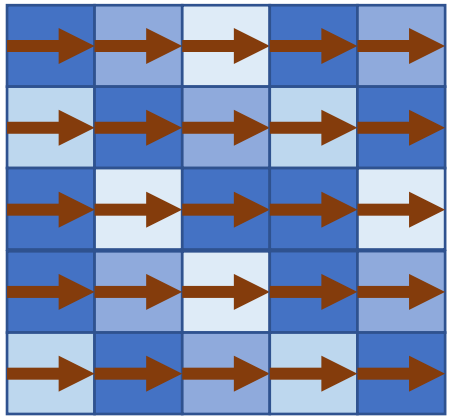


Recall - Lucas and Kanade flow

- Energy function

$$E(u,v) = \sum \left(I_x u + I_y v + I_t\right)^2$$

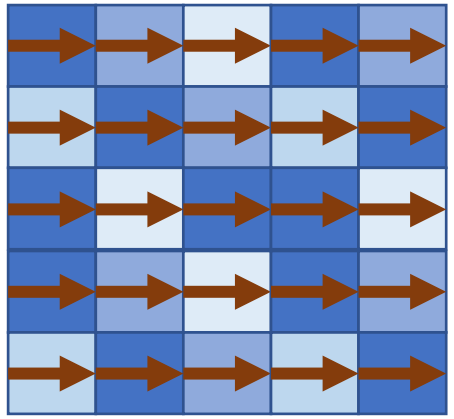
- We want to minimize this
  - $\frac{\partial E}{\partial u} = \sum 2I_x\left(I_x u + I_y v + I_t\right) = 0$
  - $\frac{\partial E}{\partial v} = \sum 2I_y\left(I_x u + I_y v + I_t\right) = 0$
- Assume the flow field is smooth locally
- Pixel neighbors have the same (u,v)



Recall - Lucas and Kanade flow (Least Square)

- Assume the flow field is smooth locally
- Pixel neighbors have the same (u,v)
- We use 5x5 windows (25 equations)

$$
\begin{aligned}
I_{x1}u + I_{y1}v &= -I_{t1} \\
I_{x2}u + I_{y2}v &= -I_{t2} \\
&\vdots \\
I_{x25}u + I_{y25}v &= -I_{t25}
\end{aligned}
$$



$$
\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y1} \\ \vdots & \vdots \\ I_{x25} & I_{y25} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{t25} \end{bmatrix}
$$

25x2 2x1 25x1

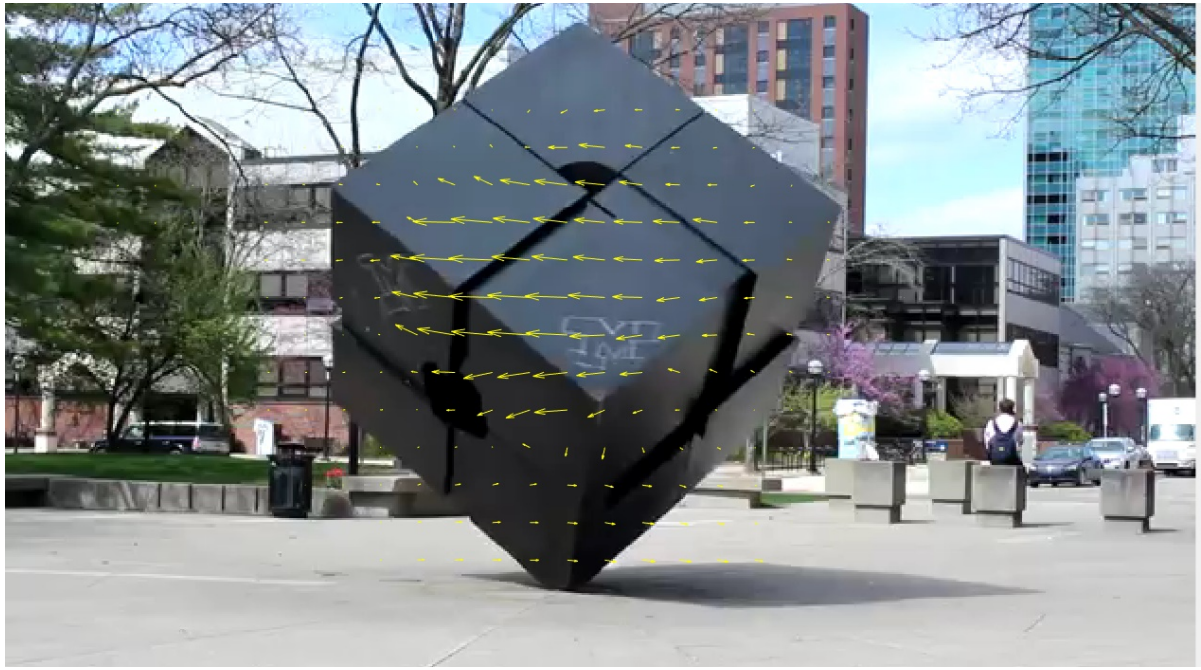
Lucas Kanade Matlab Example

- Compute Derivatives

```
fx = conv2(im1,[-1 1; -1 1], 'valid'); % partial on x
fy = conv2(im1, [-1 -1; 1 1], 'valid'); % partial on y
ft = conv2(im1, ones(2), 'valid') + conv2(im2, -ones(2), 'valid'); % partial on t
```

- Define the local windows

```
for i = w+1:size(Ix_m,1)-w
   for j = w+1:size(Ix_m,2)-w
      % Copy Ix, Iy, It to the windows
      Ix = Ix_m(i-w:i+w, j-w:j+w);
      Iy = Iy_m(i-w:i+w, j-w:j+w);
      It = It_m(i-w:i+w, j-w:j+w);

      Ix = Ix(:);
      Iy = Iy(:);
      b = -It(:); % This defines b

      A = [Ix Iy]; % This defines a
      vel = pinv(A)*b; % vel = pesudo inverse(a)*b

      u(i,j)=Vel(1);
      v(i,j)=Vel(2);
   end;
end;
```



Today’s Agenda

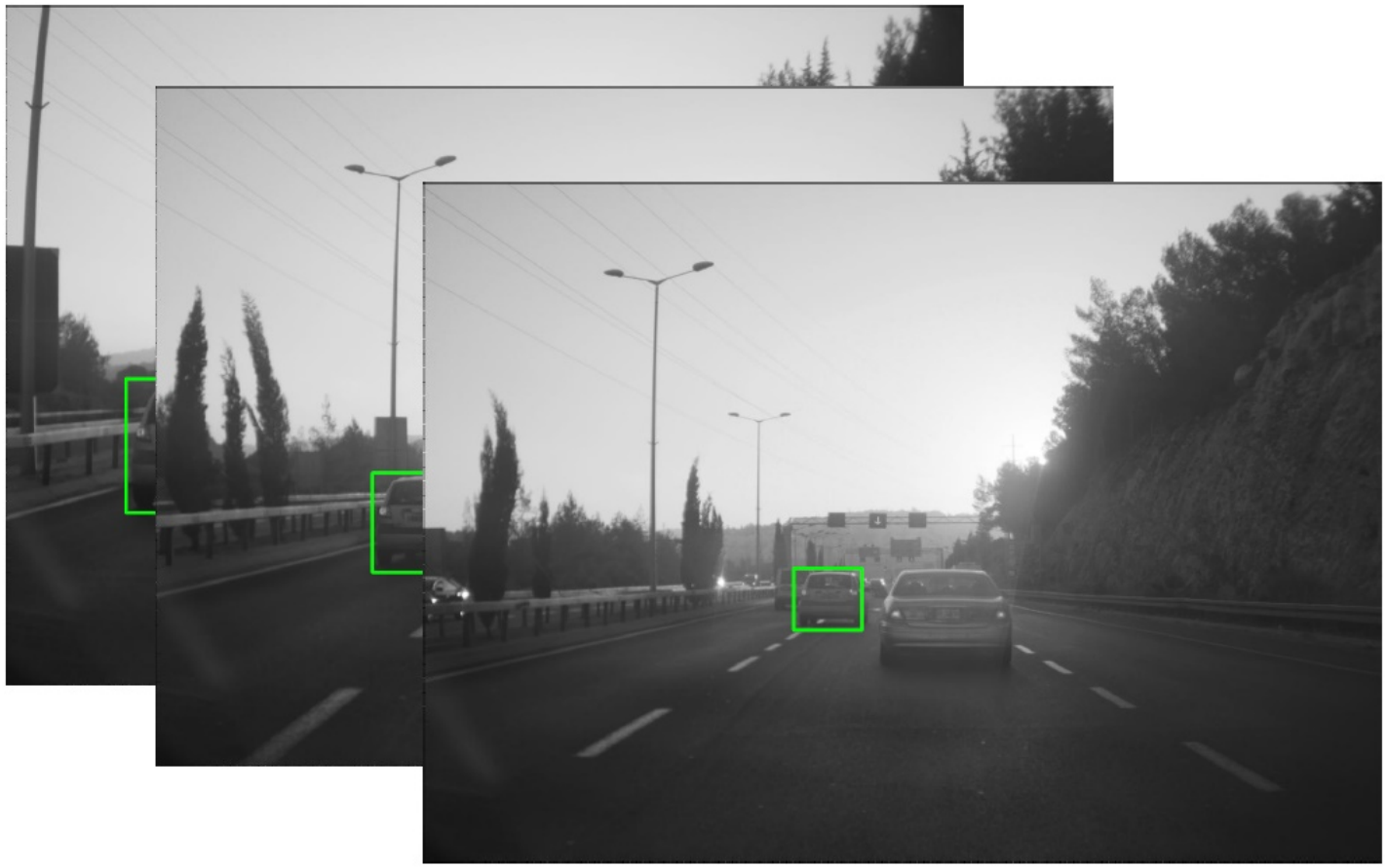
- Object Tracking
- Lucas Kanade Tomasi Feature Tracker
- Mean Shift Tracking
- Cam Shift Tracking

Tracking

Motivation

- **Safety Application based on single**
- **forward looking camera:**
- **• Lane Detection**
- - Lane Departure Warning (LDW)
- - Lane Keeping and Support
- **• Vehicle Detection**
- - Forward Collision Warning (FCW)
- - Headway Monitoring and Warning
- - Adaptive Cruise Control (ACC)
- - Traffic Jam Assistant
- - Emergency Braking (AEB)
- **• Pedestrian Detection**
- - Pedestrian Collision Warning (PCW)
- - Pedestrian Emergency Braking

Detect … Detect and Detect…



Or Tracking?

- Once the target is detected on the first frame, it should be easier to find it in the consecutive frames. This is object tracking

Template

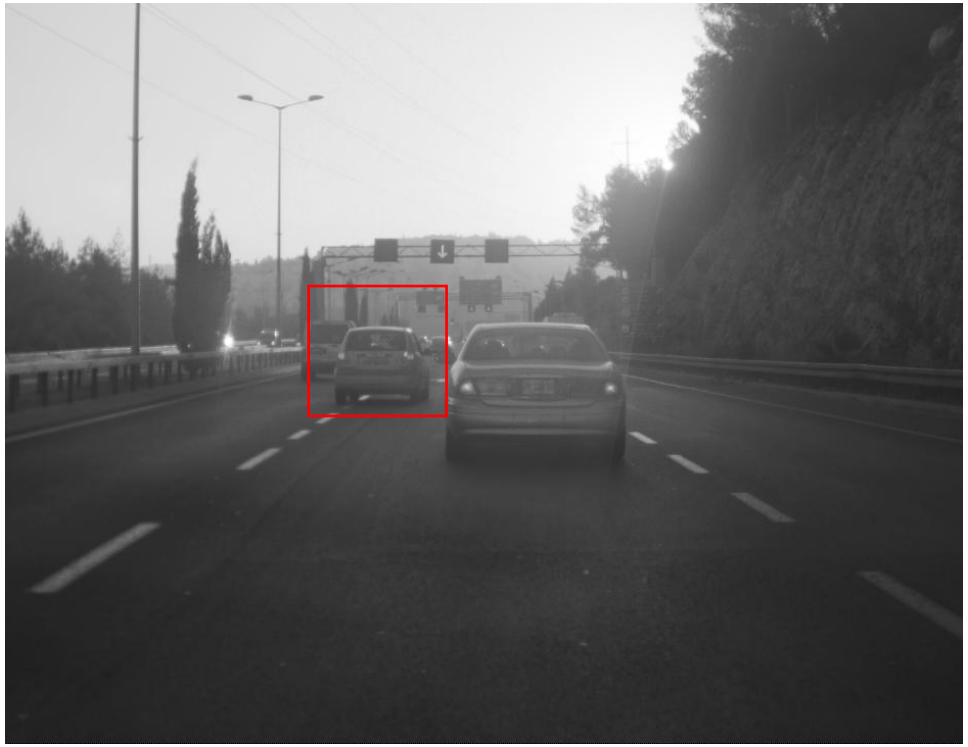


Image frames

Approaches to object tracking

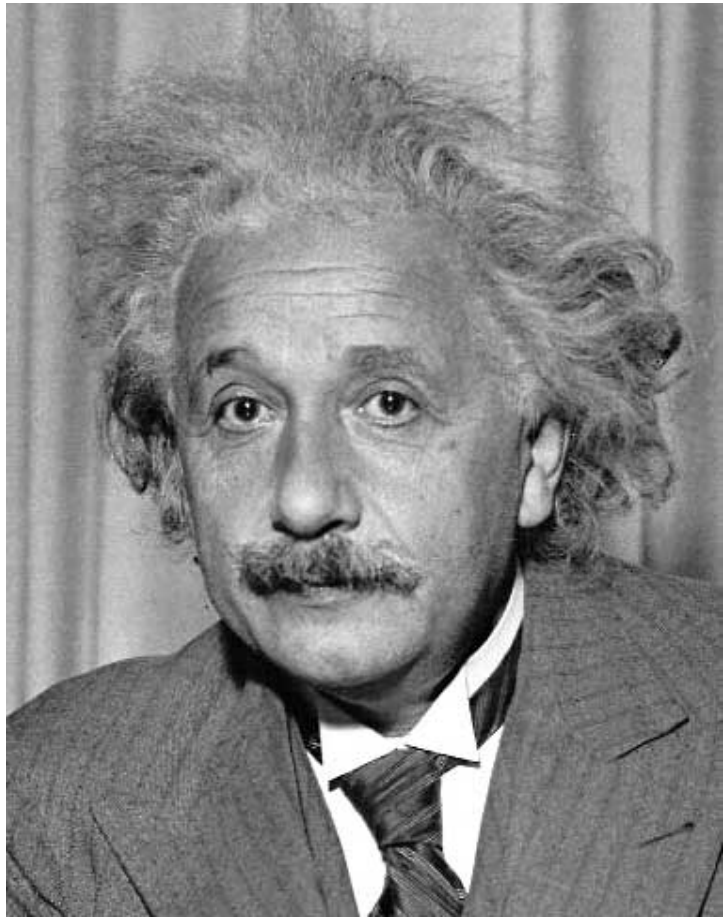
- Motion model (translation, similarily, affine, non-rigid, …)
- Image representation (gray/color pixel, edge image, histogram, HOG, wavelet...)
- Distance metric (L1, L2, normalized correlation, Chi-Squared, …)
- Method of optimization (gradient descent, naive search, combinatoric search...)
- What is tracked: whole object or selected features

Distance Metric

- Goal: To find  in the image
- Assume translation only, no scale change or rotation

- Scan over the entire image.

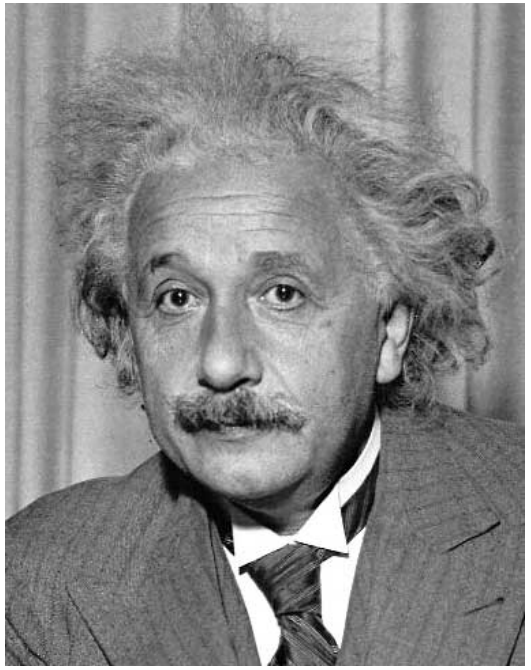
- Distance Metric
  - Correlation
  - Zero-mean correlation
  - Sum of Square Difference
  - Normalized Cross-Correlation



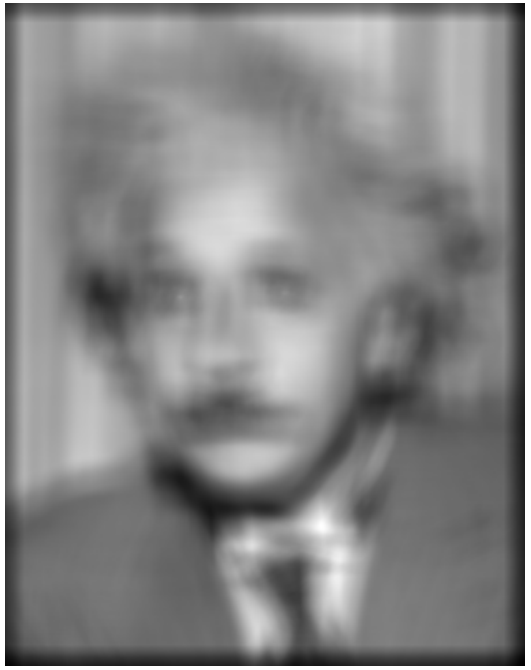
Matching with filters

- Goal: To find  in the image
- Method 0: filter the image with eye patch
  - $h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$

f=image
f=filter



Input



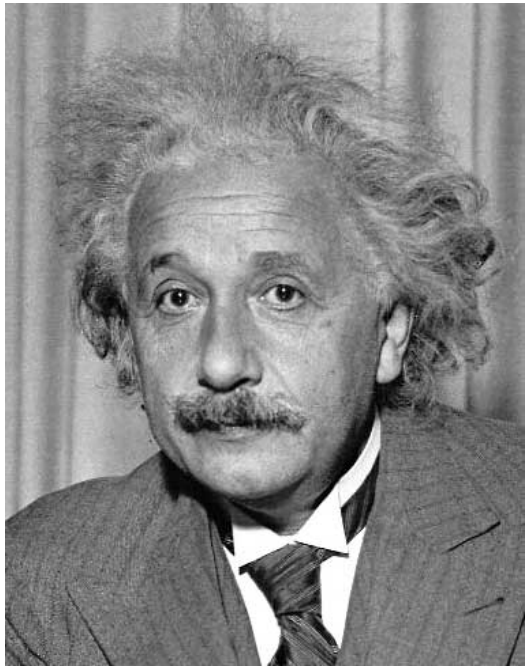
Filtered Image

What went wrong?

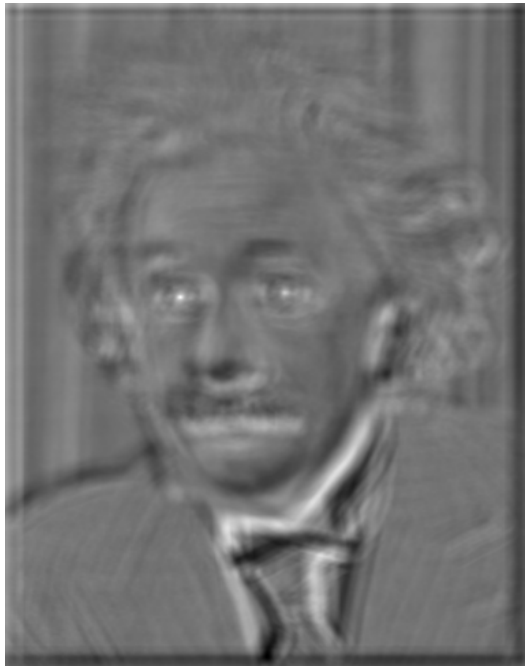
Matching with filters

- Goal: To find  in the image
- Method 1: filter the image with zero-mean eye patch
  - $h[m,n] = \sum_{k,l}(f[k,l] - \bar{f})(g[m+k,n+l])$

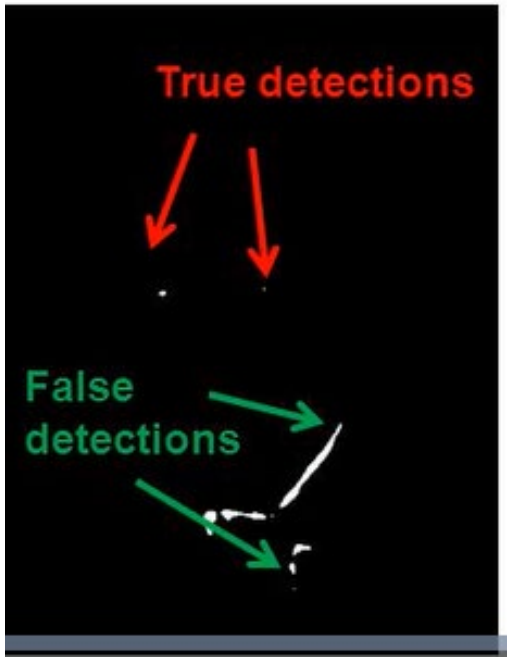
Mean of f



Input



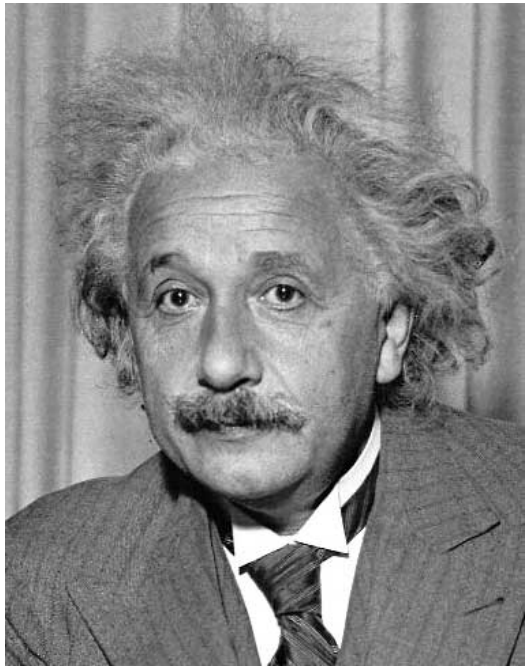
Filtered Image



Thresholded Image

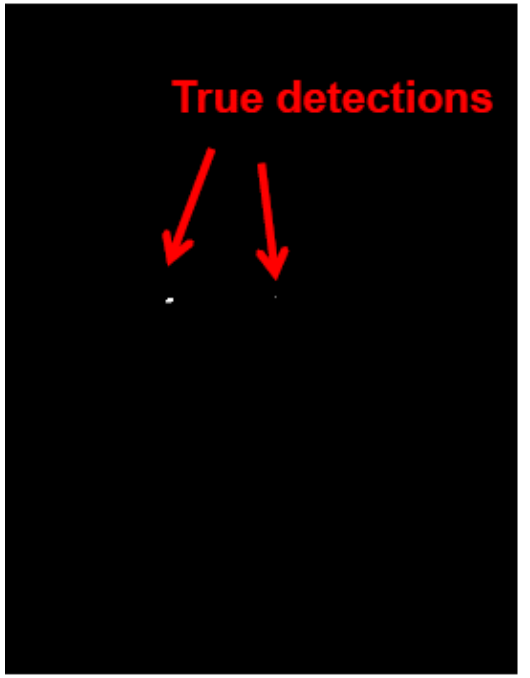
Matching with filters

- Goal: To find  in the image
- Method 2: SSD
  - $h[m,n] = \sum_{k,l}(g[k,l] - f[m+k,n+l])^2$



Input

Filtered Image

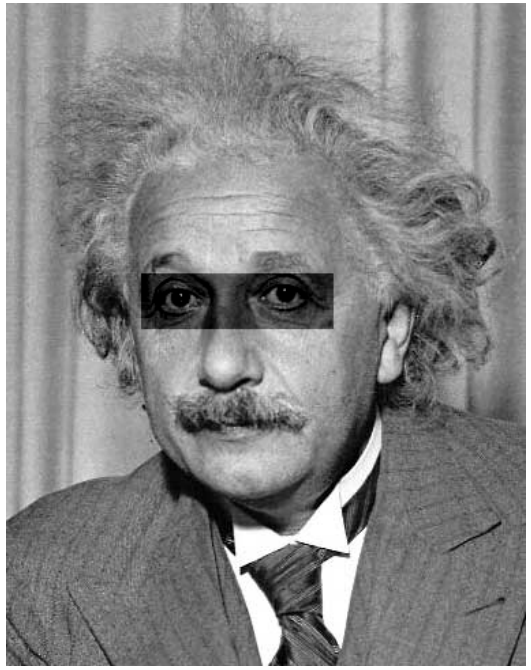


Thresholded Image

Matching with filters

- Goal: To find  in the image
- Method 2: SSD
  - $h[m,n] = \sum_{k,l}(g[k,l] - f[m+k,n+l])^2$

What's the potential downside of SSD?



Input

Filtered Image

Matching with filters

- Goal: To find  in the image
- Method 3: Normalized cross-correlation

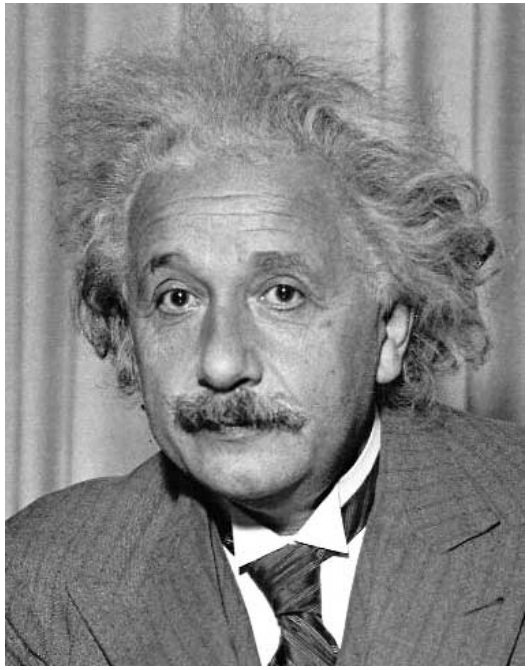
Mean template

Mean Image Patch

- $$h[m,n] = \frac{\sum_{k,l}(g[k,l]-\bar{g})(f[m-k,n-l]-\overline{f_{m,n}})}{(\sum_{k,l}(g[k,l]-\bar{g})^2\sum_{k,l}(f[m-k,n-l]-\overline{f_{m,n}})^2)^{0.5}}$$

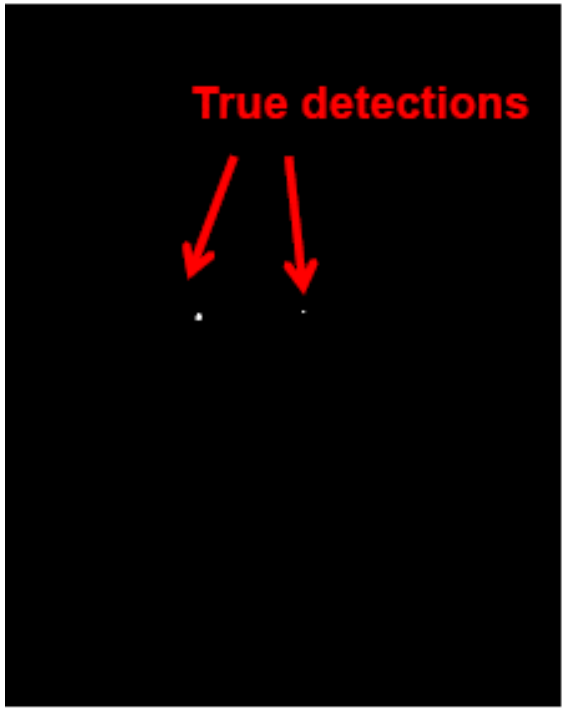
Matching with filters

- Goal: To find  in the image
- Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Example in MATLAB

```
clear;
close all;
clc;

I = imread('frame00303.jpg');
imshow(I);
T = imread('car_template.jpg');
imshow(T);

%convert it to gray
I_Grey = rgb2gray(I);
T_Grey = rgb2gray(T);

I_NCC = normxcorr2(T_Grey,I_Grey);

% This defines the position of the tracked template
[ypeak, xpeak] = find(I_NCC==max(I_NCC(:)));

yoffSet = ypeak-size(T,1);
xoffSet = xpeak-size(T,2);

%Display the matched area.
figure
imshow(I);
imrect(gca, [xoffSet+1, yoffSet+1, size(T,2), size(T,1)]);
```

Template T

Image Frame I

Search vs. Gradient Descent

- Searching:
  - Pros: Free choice of representation, distance metric; no need for good initial guess
  - Cons: expensive when searching over complex motion models (scale, rotation, affine)

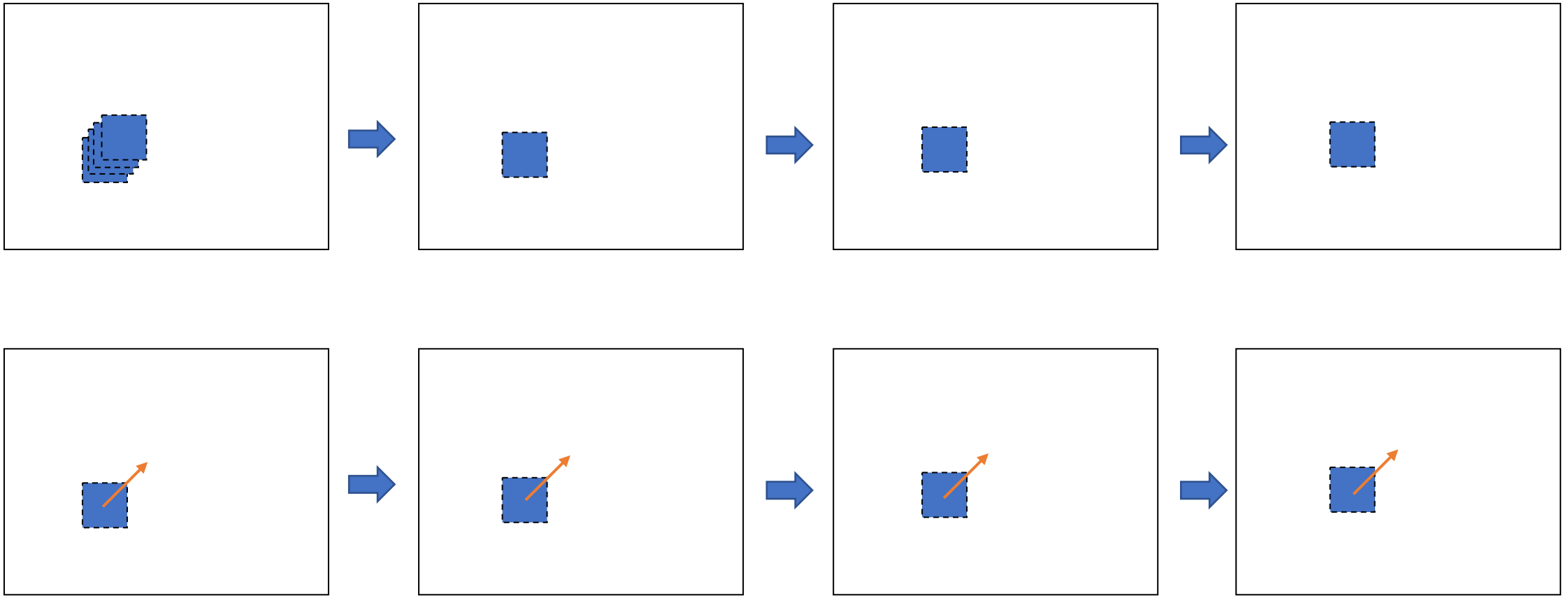
- Why Tracking instead of detecting?
  - Tracking is faster
  - Tracking can maintain the object identity
  - Detection sometimes fail when occlusion occurs

- If we have a good guess, can we do something cheaper?
  - Gradient Descent

Today’s Agenda

- Object Tracking
- Lucas Kanade Tomasi Feature Tracker
- Mean Shift Tracking

Kanade Lucas Tomasi Tracker



Before we start, review Warping functions

Translation

$$x' = x + p_1$$
$$y' = y + p_2$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} x + p_1 \\ y + p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_1 \\ 0 & 1 & p_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Parameters for the transformation

$$\mathbf{p} = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_N}\}$$

Warped Image

$$I(\mathbf{x}') = I(W(\mathbf{x};\mathbf{p}))$$

Rigid Body (Rotation + Translation)

$$x' = xcos\theta - ysin\theta + p_1$$
$$y' = xsin\theta + ycos\theta + p_2$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} cos\theta & -sin\theta & p_1 \\ sin\theta & cos\theta & p_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Review Warping functions

Affine

$$x' = a_1 x + a_2 y + b_1$$
$$y' = c_1 x + c_2 y + b_2$$

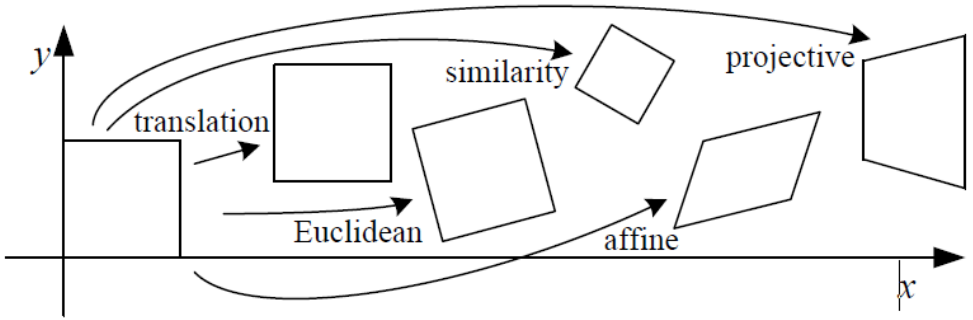
$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ c_1 & c_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective

$$x' = \frac{a_1 x + a_2 y + b_1}{c_1 x + c_2 y + 1}$$
$$y' = \frac{a_3 x + a_4 y + b_2}{c_1 x + c_2 y + 1}$$

$$W(\mathbf{x};\mathbf{p}) = [H]_{3X3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Summary of wrapping Functions



Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times3}$	2	orientation	
rigid (Euclidean)	$\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times3}$	3	lengths	
similarity	$\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times3}$	4	angles	
affine	$\left[\, \boldsymbol{A} \,\right]_{2\times3}$	6	parallelism	
projective	$\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times3}$	8	straight lines	

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} scos\theta & -ssin\theta & t_x \\ ssin\theta & scos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ c_1 & c_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Kanade Lucas Tomasi Tracker

- Key assumptions:
  - **Brightness constancy:** projection of the same point looks the same in every frame (uses SSD as metric)
  - **Small motion:** points do not move very far (from guessed location)
  - **Spatial coherence:** points move in some coherent way (according to some parametric motion model)
- For this example, assume whole object just translates in (u,v)

Template T

Image Frame I

Simple KLT Algorithm (Feature Point)

- Detect interest points (Harris Corners) in the first image
- For each interest point, compute the motion (translation or affine) between consecutive frames
- Link motion vectors in successive frames to get a track for each point
- Introduce new interest points for each 10~15 frames
- Track new and old interest points.

IEEE Conference on Computer Vision and Pattern Recognition (CVPR94) Seattle, June 1994

Good Features to Track

Jianbo Shi
Computer Science Department
Cornell University
Ithaca, NY 14853

Carlo Tomasi
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

No feature-based vision system can work unless good features can be identified and tracked from frame to frame. Although tracking itself is by and large a solved problem, selecting features that can be tracked well and correspond to physical points in the world is still hard. We propose a feature selection criterion that is optimal by construction because it is based on how the tracker works, and a feature monitoring method that can detect occlusions, disocclusions, and features that do not correspond to points in the world. These methods are based on a new tracking algorithm that extends previous Newton-Raphson style search methods to work under affine image transformations. We test performance with several simulations and experiments.

even good features can become occluded, and trackers often blissfully drift away from their original target when this occurs. No feature-based vision system can be claimed to really work until these issues have been settled.

In this paper we show how to monitor the quality of image features during tracking by using a measure of feature *dissimilarity* that quantifies the change of appearance of a feature between the first and the current frame. The idea is straightforward: dissimilarity is the feature's rms residue between the first and the current frame, and when dissimilarity grows too large the feature should be abandoned. However, in this paper we make two main contributions to this problem. First, we provide experimental evidence that pure translation is not an adequate model for image motion when measuring dissimilarity, but affine image changes, that is, linear warping and translation, are adequate. Sec-

Lucas Kanade Tomasi Tracking

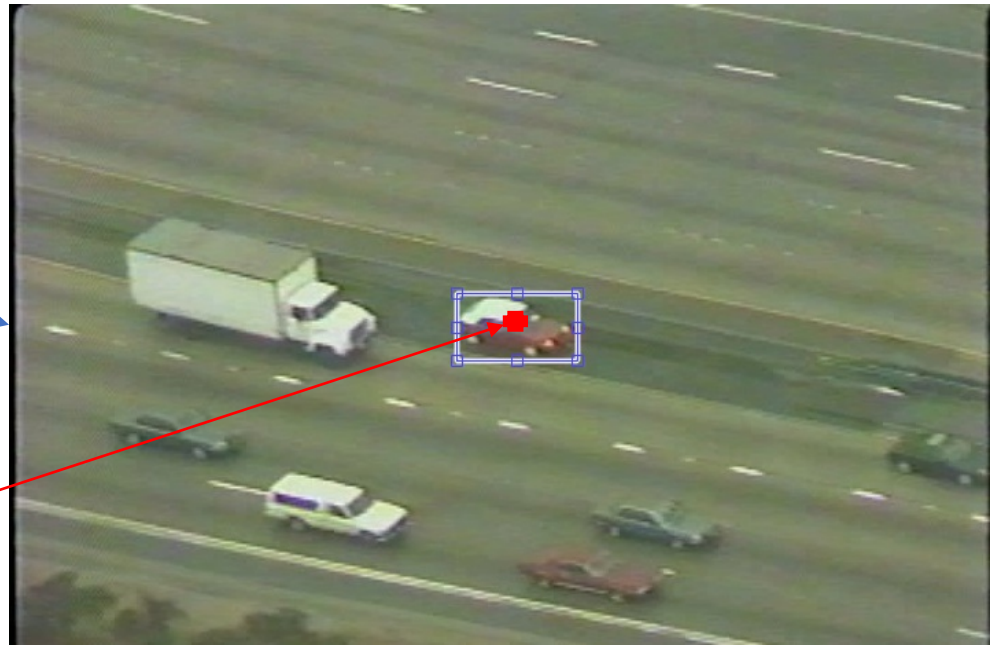
- Traditional Lucas-Kanade flow estimation works on small patches, such as corner like feature (5x5 windows) to compute the optical flow.

- Also we can use the same approach on a larger window around the object being tracked.

Basic LKT Derivation for Templates

- Energy Function for LK flow
  - $E(u,v) = \sum \left[\overbrace{I(x+u, y+v)}^{shifted\ intensity} - \overbrace{I(x,y)}^{intensity} \right]^2$
- In template tracking
  - $E(u,v) = \sum \left[\overbrace{I(x+u, y+v)}^{Image\ frame} - \overbrace{T(x,y)}^{Template} \right]^2$

Template T



current Frame I

(u,v) =hypothesized location of template in current frame

Problem with the template tracking

- Assumption of constant flow (pure translation) for all pixel in a larger window is unreasonable for long period of time.

- But one can easily generalize LK approach to other 2D parametric motion models (like affine or projective) by using "*warp*" function

Minimization Function

Start with SSD between image I and template T

$$E(u,v) = \sum [I(x+u, y+v) - T(x,y)]^2$$

E can be expressed by

$$\sum [I(W(\mathrm{x},\mathrm{y});\mathbf{p})) - T(x,y)]^2$$

We want to minimize it

$$\min_{\mathbf{p}} \sum_{\mathbf{x}} [I(W(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2 \qquad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

The Step-by-Step Derivation

We want to minimize it

$$\min_{\mathbf{p}} \sum_{\mathbf{x}} [I(W(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$$

Assume we have a good estimation of $\mathbf{p}$

$$\min_{\mathbf{p}} \sum_{\mathbf{x}} [I(W(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x})]^2$$

$\Delta\mathbf{p}$ is a small incremental change

The Step-by-Step Derivation

To derivate the above equation we applied first order Taylor expansion and

$$f(x_0 + \Delta x) \approx f(x_0) + \left.\frac{df(x)}{dx}\right|_{x=x_0} \Delta x$$

We have

$$I(W(\mathrm{x},\mathrm{y});\mathbf{p}+\Delta\mathbf{p})) \approx I(W(\mathrm{x},\mathrm{y});\mathbf{p})) + \frac{\partial I(W(x,y);\mathbf{p})}{\partial \mathbf{P}} \Delta\mathbf{p}$$

By chain rule

$f(x,y)$ $x = g(t)$ $y = h(t)$

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} \qquad \frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial g(t)}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial h(t)}{\partial t}$$

Taylor Expansion

- Consider single variable *P*

$$I(W(\mathrm{x},\mathrm{y});\hat{P}+\Delta P)) \approx I(W(\mathrm{x},\mathrm{y});P)) + \frac{\partial I(W(x,y);P)}{\partial P}\Delta P$$

Recall

$$\mathbf{x}' = \mathrm{W}(\mathbf{x},\mathbf{p})$$

$$\frac{\partial I(W(x,y);P)}{\partial P} = \frac{\partial I(W(x,y);P)}{\partial \mathbf{x}'}\frac{\partial W((x,y);P)}{\partial P}$$

We have

$$I(W(\mathrm{x},\mathrm{y});\hat{P})) + \left.\frac{\partial I}{\partial P}\right|_{P_0}\Delta P = I(W(\mathrm{x},\mathrm{y});\hat{P})) + \left.\left[\frac{\partial I}{\partial x}\frac{\partial W_x}{\partial P} + \frac{\partial I}{\partial y}\frac{\partial W_y}{\partial P}\right]\right|_{\hat{P}}\Delta P$$

Taylor Expansion vector function

- First order Taylor Expansion for scalar function

$$f(x_0 + \Delta x) \approx f(x_0) + \left.\frac{df(x)}{dx}\right|_{x=x_0} \Delta x$$

- For vector function

$$F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}),\ldots, f_m(\mathbf{x})) \text{ where } \mathrm{x} = [x_1, x_2, \ldots, x_n]$$

we have

$$f_1(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f_1(\mathbf{x}_0) + \left.\frac{\partial f_1(\mathbf{x})}{\partial x_1}\right|_{x=x_0} \Delta x_1 + \cdots + \left.\frac{\partial f_1(\mathbf{x})}{\partial x_1}\right|_{x=x_0} \Delta x_n$$

$$f_m(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f_m(\mathbf{x}_0) + \left.\frac{\partial f_m(\mathbf{x})}{\partial x_1}\right|_{x=x_0} \Delta x_1 + \cdots + \left.\frac{\partial f_m(\mathbf{x})}{\partial x_1}\right|_{x=x_0} \Delta x_n$$

Taylor Expansion of Vector Function

- $$F(\mathbf{x}_0 + \Delta\mathbf{x}) \approx F(\mathbf{x}_0) + \left[\begin{matrix} \frac{\partial f_1}{\partial x_1} & \cdots & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \end{matrix}\right]_{\mathbf{x}=\mathbf{x_0}} \Delta\mathbf{x}$$

Jacobian

- Suppose we have vector function

$$F(x_1, x_2, \dots, x_n) = (f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_n\ (x_1, x_2, \dots, x_n))$$

- Derivative of vector Function

$$J(F) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \dots & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Taylor Expansion

- Consider Multivariable Taylor Series Expansion

$$f(x,y) \approx f(a,b) + f_x(a,b)(x-a) + f_y(a,b)(y-b)$$

- Let's consider a warping functions of n variables $P_1, P_2 \ldots P_n$

$$W_x = W_x(P_1, P_2 \ldots P_n)$$
$$W_y = W_y(P_1, P_2 \ldots P_n)$$

Taylor Expansion

- Let's consider a warping functions of n variables $P_1, P_2 \dots P_n$

$$W_x = W_x(P_1, P_2 \dots P_n)$$
$$W_y = W_y(P_1, P_2 \dots P_n)$$

$$I\left(W(\mathrm{x},\mathrm{y}); \widehat{P_1} + \Delta P_1, \widehat{P_2} + \Delta P_2 \dots \widehat{P_n} + \Delta P_n\right) \approx$$

$$I\left(W(\mathrm{x},\mathrm{y}); \widehat{P_1}, \widehat{P_2} \dots \widehat{P_n}\right) + \left[\frac{\partial I}{\partial x}\frac{\partial W_x}{\partial P_1} + \frac{\partial I}{\partial y}\frac{\partial W_y}{\partial P_1}\right]\Bigg|_{\widehat{P_1}} \Delta P_1 + \left[\frac{\partial I}{\partial x}\frac{\partial W_x}{\partial P_2} + \frac{\partial I}{\partial y}\frac{\partial W_y}{\partial P_2}\right]\Bigg|_{\widehat{P_2}} \Delta P_2 + \cdots + \left[\frac{\partial I}{\partial x}\frac{\partial W_x}{\partial P_n} + \frac{\partial I}{\partial y}\frac{\partial W_y}{\partial P_n}\right]\Bigg|_{\widehat{P_n}} \Delta P_n$$

Derivation

$$\left[\frac{\partial I}{\partial x}\frac{\partial W_x}{\partial P_i}+\frac{\partial I}{\partial y}\frac{\partial W_y}{\partial P_i}\right]\Bigg|_{\widehat{P_i}}\Delta P_i=\left[\frac{\partial I}{\partial x}\quad\frac{\partial I}{\partial y}\right]\left[\begin{matrix}\frac{\partial W_x}{\partial P_i}\\ \frac{\partial W_y}{\partial P_i}\end{matrix}\right]\Bigg|_{\widehat{P_i}}\Delta P_i$$

Therefore we have

$$I\left(W(\mathrm{x},\mathrm{y});\widehat{P_1}+\Delta P_1,\widehat{P_2}+\Delta P_2\ldots\widehat{P_n}+\Delta P_n\right)\approx I\left(W(\mathrm{x},\mathrm{y});\widehat{P_1},\widehat{P_2}\ldots\widehat{P_n}\right)+$$

$$\left[\frac{\partial I}{\partial x}\quad\frac{\partial I}{\partial y}\right]\left[\begin{matrix}\frac{\partial W_x}{\partial P_1}\\ \frac{\partial W_y}{\partial P_1}\end{matrix}\right]\Bigg|_{\widehat{P_1}}\Delta P_1+\left[\frac{\partial I}{\partial x}\quad\frac{\partial I}{\partial y}\right]\left[\begin{matrix}\frac{\partial W_x}{\partial P_2}\\ \frac{\partial W_y}{\partial P_2}\end{matrix}\right]\Bigg|_{\widehat{P_2}}\Delta P_2+\cdots+\left[\frac{\partial I}{\partial x}\quad\frac{\partial I}{\partial y}\right]\left[\begin{matrix}\frac{\partial W_x}{\partial P_n}\\ \frac{\partial W_y}{\partial P_n}\end{matrix}\right]\Bigg|_{\widehat{P_n}}\Delta P_n$$

Derivation

- $I\left(W(\mathrm{x},\mathrm{y});\widehat{P_1}+\Delta P_1,\widehat{P_2}+\Delta P_2\ldots\widehat{P_n}+\Delta P_n\right)\approx$

$$I\left(W(\mathrm{x},\mathrm{y});\widehat{P_1},\widehat{P_2}\ldots\widehat{P_n}\right)+\begin{bmatrix}\frac{\partial I}{\partial x} & \frac{\partial I}{\partial y}\end{bmatrix}\begin{bmatrix}\frac{\partial W_x}{\partial P_1} & \frac{\partial W_x}{\partial P_2} & \cdots & \frac{\partial W_x}{\partial P_n}\\ \frac{\partial W_y}{\partial P_1} & \frac{\partial W_y}{\partial P_2} & \cdots & \frac{\partial W_y}{\partial P_n}\end{bmatrix}\begin{bmatrix}\Delta P_1\\ \Delta P_2\\ \vdots\\ \Delta P_n\end{bmatrix}$$

Gradient (∇I)

Jacobian($\frac{\partial W}{\partial P}$)

Increment Parameters($\Delta \boldsymbol{P}$)

Rewriting

$$I(W(\mathrm{x},\mathrm{y});\boldsymbol{P}+\Delta\boldsymbol{P})\approx I(W(\mathrm{x},\mathrm{y});\boldsymbol{P})+\nabla I\frac{\partial W}{\partial P}\Delta\boldsymbol{P}$$

Example: Jacobian of Affine warp

- Let

$$W\big((\mathrm{x},\mathrm{y});\boldsymbol{P}\big) = [W_x \quad W_y]$$

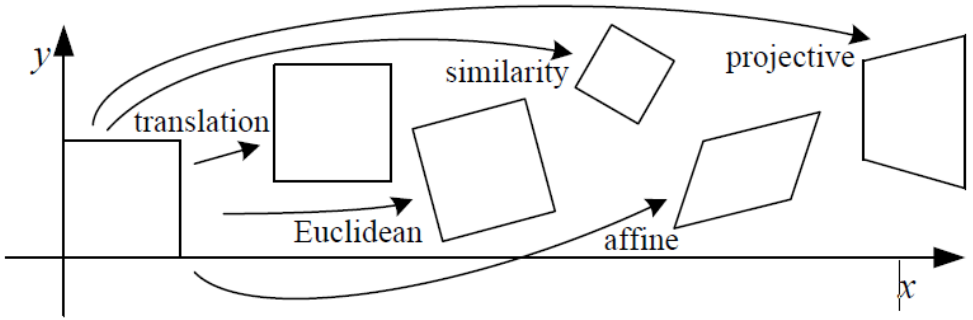
$$\frac{\partial W}{\partial \boldsymbol{P}} = \begin{bmatrix} \frac{\partial W_x}{\partial P_1} & \frac{\partial W_x}{\partial P_2} & \cdots & \frac{\partial W_x}{\partial P_n} \\ \frac{\partial W_y}{\partial P_1} & \frac{\partial W_y}{\partial P_2} & \cdots & \frac{\partial W_y}{\partial P_n} \end{bmatrix}$$

- Example: For Affine transform, we have 6 parameters in warp function

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ c_1 & c_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\frac{\partial W}{\partial \boldsymbol{P}} = \frac{\partial \begin{bmatrix} a_1x+a_2y+b_1 \\ c_1x+c_2y+b_2 \end{bmatrix}}{\partial \boldsymbol{P}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}$$

Summary of wrapping Functions



Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times3}$	2	orientation	
rigid (Euclidean)	$\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times3}$	3	lengths	
similarity	$\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times3}$	4	angles	
affine	$\left[\, \boldsymbol{A} \,\right]_{2\times3}$	6	parallelism	
projective	$\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times3}$	8	straight lines	

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} scos\theta & -ssin\theta & t_x \\ ssin\theta & scos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ c_1 & c_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Derivative of warping functions

Translation

$$x' = x + b_1$$
$$y' = y + b_2$$
$$W(\mathbf{x};\mathbf{p}) = (\mathrm{x} + b_1, \mathrm{y} + b_2)$$

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Affine

$$x' = a_1 x + a_2 y + b_1$$
$$y' = c_1 x + c_2 y + b_2$$
$$W(\mathbf{x};\mathbf{p}) = (a_1 x + a_2 y + b_1, c_1 x + c_2 y + b_2)$$

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}$$

Rigid Body (Rotation + Translation)

$$x' = x\cos\theta - y\sin\theta + b_1$$
$$y' = x\sin\theta + y\cos\theta + b_2$$
$$W(\mathbf{x};\mathbf{p}) = (x\cos\theta - y\sin\theta + b_1, x\sin\theta + y\cos\theta + b_2)$$

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 & x\sin\theta - y\cos\theta \\ 0 & 1 & x\cos\theta - y\sin\theta \end{bmatrix}$$

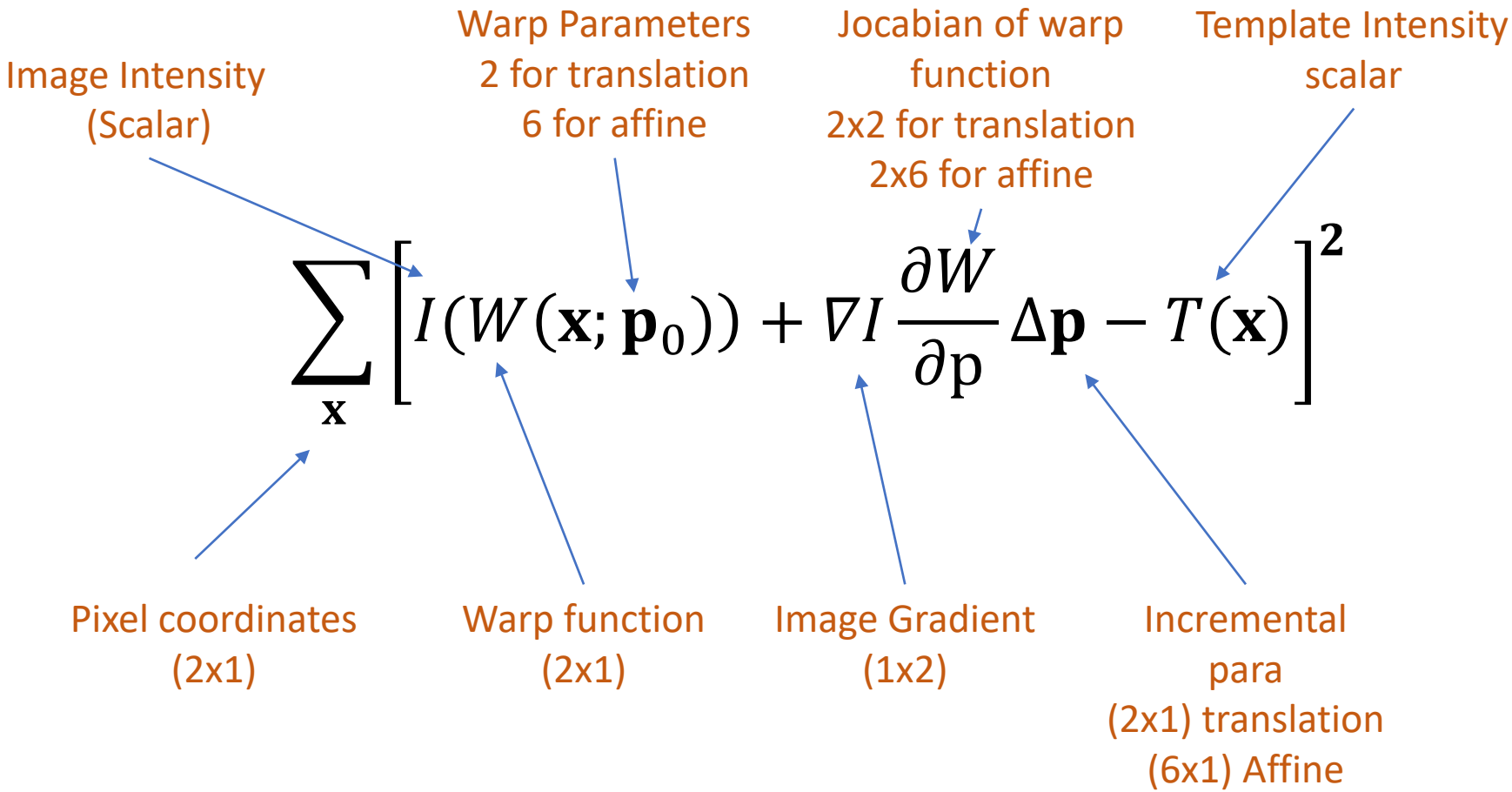
Summary of Jacobian

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Parameters $\boldsymbol{p}$	Jacobian $\boldsymbol{J}$
(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Source: R. Szeliski

What you need to know



Summary of process - Find Alignment

- Goal: Given Template $T(\mathbf{x})$, find $\mathbf{p}$ to minimize the energy function

$$\sum_{\mathbf{x}}[I(W(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$$

- Consider $\mathbf{p}_0 + \Delta\mathbf{p}$, $\mathbf{p}_0$ is known, find $\Delta\mathbf{p}$

$$\sum_{\mathbf{x}}[I(W(\mathbf{x};\mathbf{p}_0 + \Delta\mathbf{p})) - T(\mathbf{x})]^2$$

- By Taylor Series

$$\sum_{\mathbf{x}}\left[I(W(\mathbf{x};\mathbf{p}_0)) + \nabla I \frac{\partial W}{\partial \mathrm{p}}\Delta\mathbf{p} - T(\mathbf{x})\right]^2$$

$$\nabla I = [I_x \quad I_y]$$

Finding Alignment

- We want to minimize this

$$\sum_{\mathbf{x}} \left[I(W(\mathbf{x};\mathbf{p}_0)) + \nabla I \frac{\partial W}{\partial \mathrm{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2$$

Differentiate w.r.t. $\Delta\mathbf{p}$ and set it to zero

$$2\sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} \left[I(W(\mathbf{x};\mathbf{p}_0)) + \nabla I \frac{\partial W}{\partial \mathrm{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] = 0$$

$$\sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right] \Delta\mathbf{p} = \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}_0))]$$

Finding Alignment (Cont'd)

$$\sum_{\mathrm{x}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]\Delta\mathbf{p}=\sum_{\mathrm{x}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}}\left[T(\mathbf{x})-I(W(\mathbf{x};\mathbf{p}_0))\right]$$

$$\Delta\mathbf{p}=H^{-1}\sum_{\mathrm{x}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}}\left[T(\mathbf{x})-I(W(\mathbf{x};\mathbf{p}_0))\right]$$

- Where H is the *Hessian Matrix*

$$H=\sum_{\mathrm{x}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}}\left[\nabla I\frac{\partial W}{\partial \mathrm{p}}\right]$$

Stability of the gradient decent iterations

The gradient update:

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}_0))]$$

- When will the inversion fail?

  H is singular

- Well-conditioned
  - Both Eigenvalues have similar magnitude

Stability of gradient decent in Translation Motion case

- $H = \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}}\right]$
- For translation,

$$W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} \mathrm{x} + p_1 \\ \mathrm{y} + p_2 \end{bmatrix}$$

Therefore

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\nabla I = [I_x \quad I_y]$$

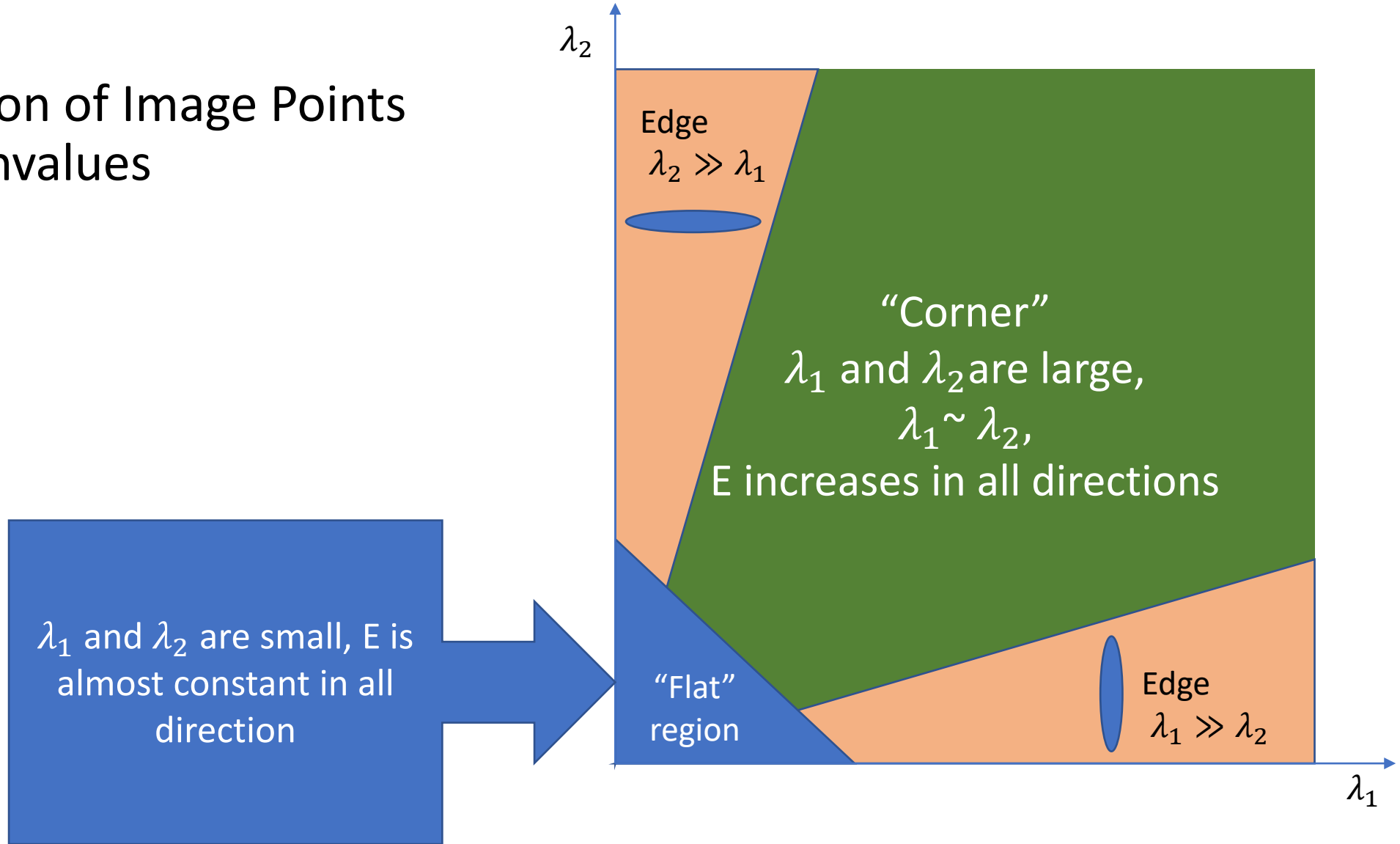
$$\nabla I \frac{\partial W}{\partial \mathrm{p}} = [I_x \quad I_y] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [I_x \quad I_y]$$

$$\left[\nabla I \frac{\partial W}{\partial \mathrm{p}}\right]^T = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

- $H = \sum_{\mathrm{x}} \begin{bmatrix} I_x \\ I_y \end{bmatrix}^{\mathrm{T}} [I_x \quad I_y]$
- $H = \sum_{\mathrm{x}} \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$
- What is this?
- ***Harris Corner Detector***

Interpreting the eignvalues

- Classification of Image Points using eigenvalues

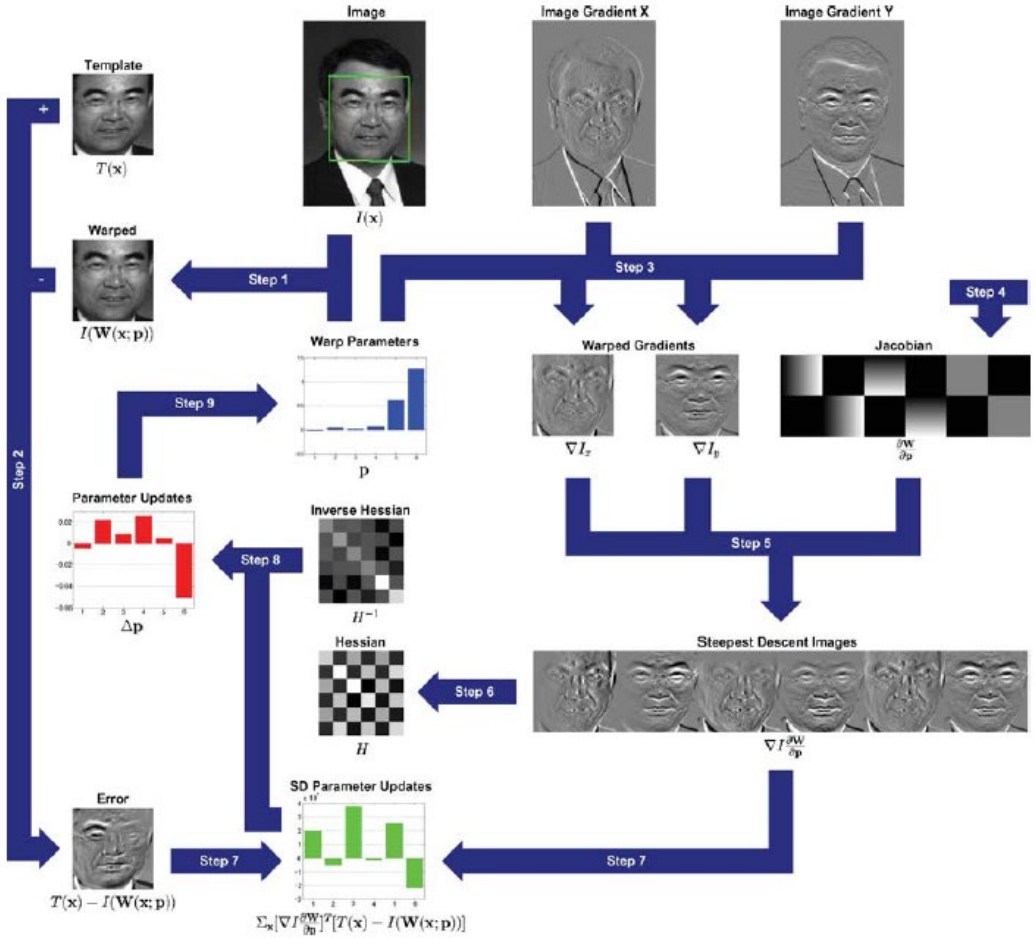


KLT Algorithm

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}))]$$

1. Warp I with $W(\mathbf{x}; \mathbf{p}))$
2. Subtract I from T $[T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}))]$
3. Compute Gradient $\nabla I(\mathbf{x}')$
4. Evaluate the Jacobian $\frac{\partial W}{\partial \mathrm{p}}$
5. Compute steepest decent $\nabla I \frac{\partial W}{\partial \mathrm{p}}$
6. Compute Inverse Hessian H^{-1}
7. Multiply steepest descend with error $\sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}))]$
8. Compute $\Delta \mathbf{p}$ $\Delta \mathbf{p} = H^{-1} \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}))]$
9. Update Parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$

KLT-Kanade Lucas Tomasi



1. Warp I with $W(\mathbf{x};\mathbf{p}))$
2. Subtract I from T $[T(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]$
3. Compute Gradient $\nabla I(\mathbf{x}')$
4. Evaluate the Jacobian $\frac{\partial W}{\partial \mathrm{p}}$
5. Compute steepest decent $\nabla I \frac{\partial W}{\partial \mathrm{p}}$
6. Compute Inverse Hessian H^{-1}
7. Multiply steepest descend with error $\sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]$
8. Compute $\Delta\mathbf{p}$

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W}{\partial \mathrm{p}}\right]^{\mathrm{T}} [T(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]$$

9. Update Parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

Variations of KLT (Baker et al, IJCV 2004)

- In LKT, we have

$$\frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p}_0 + \Delta \mathbf{p})) - T(\mathbf{x})]^2 = 0$$

- We can approximate as compositional wrapping

$$\frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} \left[I\left(W\left(W(\mathbf{x}; \mathbf{p}_0); \Delta \mathbf{p}\right)\right) - T(\mathbf{x}) \right]^2 = 0$$

- We wish to have $\mathbf{p}_0$ ***stationary*** w.r.t. any $\Delta \mathbf{p}$, we can rewrite the equation as

$$\frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} \left[I\left(W\left(W(\mathbf{x}; \mathbf{p}_0); -\Delta \mathbf{p}\right)\right) - T(\mathbf{x}) \right]^2$$

$$\frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} \left[I\left(W(\mathbf{x}; \mathbf{p}_0)\right) - T(W(\mathbf{x}; \Delta \mathbf{p})) \right]^2$$

Variations of KLT(Baker et al, IJCV 2004) (Cont'd)

- $\frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} \left[I\left(W(\mathbf{x}; \mathbf{p}_0)\right) - T(W(\mathbf{x};\ \Delta \mathbf{p})) \right]^2$

$$\approx \frac{\partial}{\partial \Delta \mathbf{p}} \sum_{\mathbf{x}} \left[I\left(W(\mathbf{x}; \mathbf{p}_0)\right) - T(W(\mathbf{x}; 0) - \Delta T \frac{\partial W(x;0)}{\partial p} \Delta \mathbf{p}) \right]^2$$

- We equate it to zero

$$2 \sum_{\mathrm{x}} \left[\nabla I \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} \left[I(W(\mathbf{x}; \mathbf{p}_0)) - T(\mathbf{x}) - \nabla T \frac{\partial W(\mathbf{x};0)}{\partial \mathbf{p}} \Delta \mathbf{p} \right] = 0$$

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial W(\mathbf{x}; 0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} [I(W(\mathbf{x}; \mathbf{p}_0)) - T(\mathbf{x})]$$

- Where $H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]$

What is so great about Baker's approach?

Original KLT:

$$\sum_{\mathbf{x}}[I(W(\mathbf{x};\mathbf{p}_0+\Delta\mathbf{p}))-T(\mathbf{x})]^2$$

- Linearized form

$$\sum_{\mathbf{x}}\left[I(W(\mathbf{x};\mathbf{p}_0))+\nabla I\frac{\partial W}{\partial \mathrm{p}}\Delta\mathbf{p}-T(\mathbf{x})\right]^2$$

Jacobian need to recompute each iteration

Baker's Approach:

$$\sum_{\mathbf{x}}\left[I\left(W(W(\mathbf{x};\mathbf{p}_0);\Delta\mathbf{p})\right)-T(\mathbf{x})\right]^2$$

- Linearized form

$$\sum_{\mathbf{x}}\left[I\left(W(\mathbf{x};\mathbf{p}_0)\right)-T(W(\mathbf{x};0)-\Delta T\frac{\partial W(x;0)}{\partial p}\Delta\mathbf{p})\right]^2$$

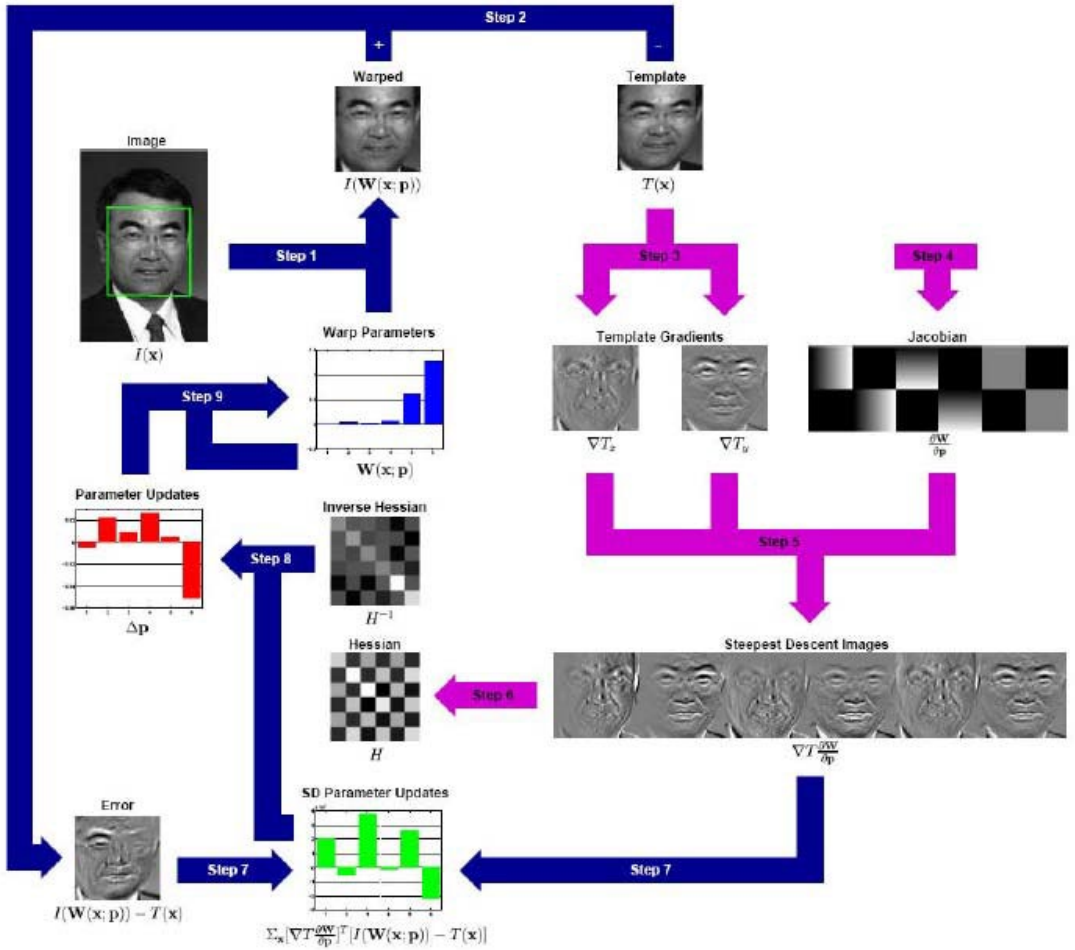
Jacobian is constant
Can be pre-computed

Modified KLT (Baker et al, IJCV 2004)

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$$

1. Warp I with $W(\mathbf{x};\mathbf{p}))$
2. Subtract T from I $\quad [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$
3. Compute Gradient ∇T (Only do once)
4. Evaluate the Jacobian $\frac{\partial W}{\partial \mathrm{p}}$ at (x;0) (Only do once)
5. Compute steepest decent $\nabla I \frac{\partial W}{\partial \mathrm{p}}$ (Only do once)
6. Compute Inverse Hessian H^{-1} (Only do once)
7. Multiply steepest descend with error $\quad \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$
8. Compute $\Delta \mathbf{p}$ $\quad \Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}} \right]^{\mathrm{T}} [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$
9. Update Parameters $\quad \mathbf{p} \rightarrow \mathbf{p} + \Delta \mathbf{p}$

Modified Lucas-Kanade (Baker et al, IJCV 2004)



1. Warp I with $W(\mathbf{x};\mathbf{p}))$
2. Subtract T from I $\quad [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$
3. Do it once
   1. Compute Gradient ∇T
   2. Evaluate the Jacobian $\frac{\partial W}{\partial \mathrm{p}}$ at (x;0)
   3. Compute steepest decent $\nabla I \frac{\partial W}{\partial \mathrm{p}}$
   4. Compute Inverse Hessian H^{-1}
4. Multiply steepest descend with error

$$\sum_{\mathbf{x}} \left[\nabla I \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}}\right]^{\mathrm{T}} [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$$

5. Compute $\Delta\mathbf{p}$

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial W(\mathbf{x};0)}{\partial \mathrm{p}}\right]^{\mathrm{T}} [I(W(\mathbf{x};\mathbf{p}_0)) - T(\mathbf{x})]$$

6. Update Parameters $\quad \mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

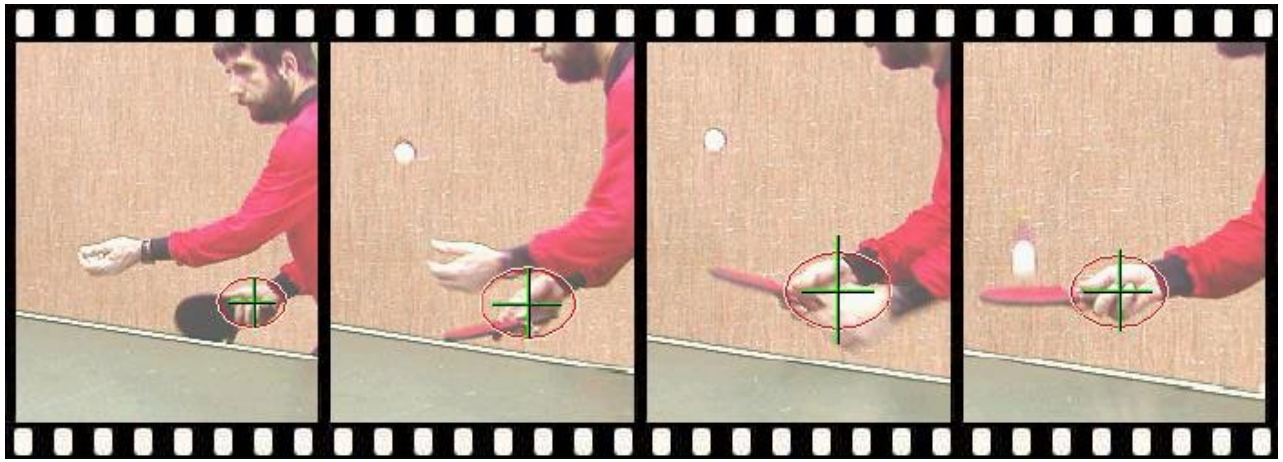
Baker *et al*, IJCV, 2004.

Today’s Agenda

- Object Tracking
- Lucas Kanade Tomasi Feature Tracker
- Mean Shift Tracking

Mean Shift Tracking

- Goal: ***Realtime*** tracking of ***non-rigid object***



IEEE CVPR 2000

Real-Time Tracking of Non-Rigid Objects using Mean Shift

Dorin Comaniciu Visvanathan Ramesh
Imaging & Visualization Department
Siemens Corporate Research
755 College Road East, Princeton, NJ 08540

Peter Meer
Electrical & Computer Engineering Department
Rutgers University
94 Brett Road, Piscataway, NJ 08855

Abstract

A new method for real-time tracking of non-rigid objects seen from a moving camera is proposed. The central computational module is based on the mean shift iterations and finds the most probable target position in the current frame. The dissimilarity between the target model (its color distribution) and the target candidates is expressed by a metric derived from the Bhattacharyya coefficient. The theoretical analysis of the approach shows that it relates to the Bayesian framework while providing a practical, fast and efficient solution. The capability of the tracker to handle in real-time partial occlusions, significant clutter, and target scale variations, is demonstrated for several image sequences.

1 Introduction

The efficient tracking of visual features in complex environments is a challenging task for the vision community. Real-time applications such as surveillance and monitoring [10], perceptual user interfaces [4], smart rooms [16, 28], and video compression [12] all require the ability to track moving objects. The computational complexity of the tracker is critical for most applications, only a small percentage of a system resources being allocated for tracking, while the rest is assigned to preprocessing stages or to high-level tasks such as recognition, trajectory interpretation, and reasoning [24].

This paper presents a new approach to the real-time tracking of non-rigid objects based on visual features such as color and/or texture, whose statistical distributions characterize the object of interest. The proposed tracking is appropriate for a large variety of objects with

2 Mean Shift Analysis

We define next the sample mean shift, introduce the iterative mean shift procedure, and present a new theorem showing the convergence for kernels with convex and monotonic profiles. For applications of the mean shift property in low level vision (filtering, segmentation) see [6].

2.1 Sample Mean Shift

Given a set $\{\mathbf{x}_i\}_{i=1\ldots n}$ of n points in the d-dimensional space R^d, the *multivariate kernel density estimate* with kernel $K(\mathbf{x})$ and window radius (bandwidth) h, computed in the point $\mathbf{x}$ is given by

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right). \quad (1)$$

The minimization of the average global error between the estimate and the true density yields the multivariate Epanechnikov kernel [25, p.139]

$$K_E(\mathbf{x}) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1-\|\mathbf{x}\|^2) & \text{if } \|\mathbf{x}\|<1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

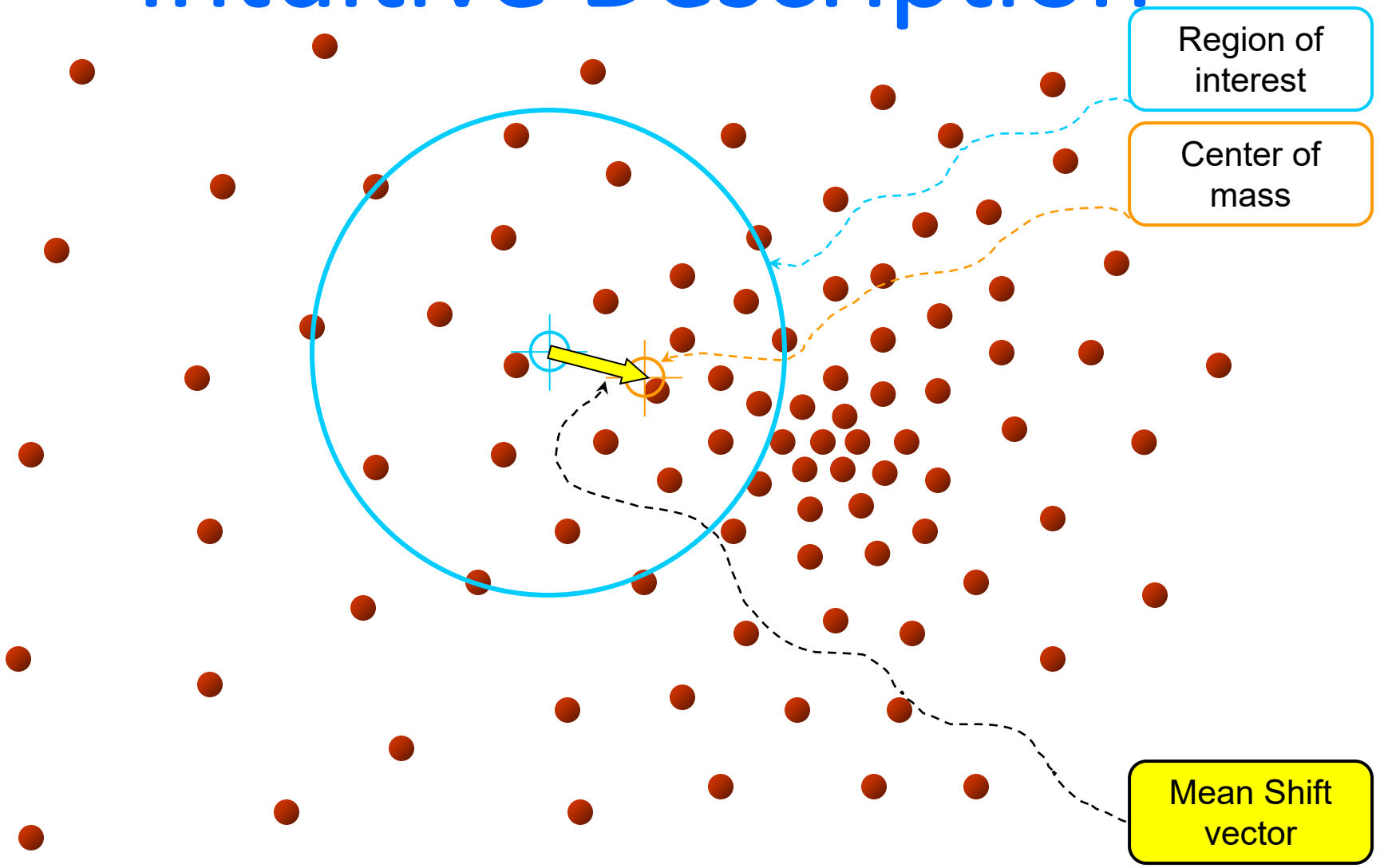
where c_d is the volume of the unit d-dimensional sphere. Another commonly used kernel is the multivariate normal

$$K_N(\mathbf{x}) = (2\pi)^{-d/2}\exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right). \quad (3)$$

Let us introduce the *profile* of a kernel K as a function $k:[0,\infty)\rightarrow R$ such that $K(\mathbf{x}) = k(\|\mathbf{x}\|^2)$. For example, according to (2) the Epanechnikov profile is

$$k_E(x) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1-x) & \text{if } x<1 \end{cases} \quad (4)$$

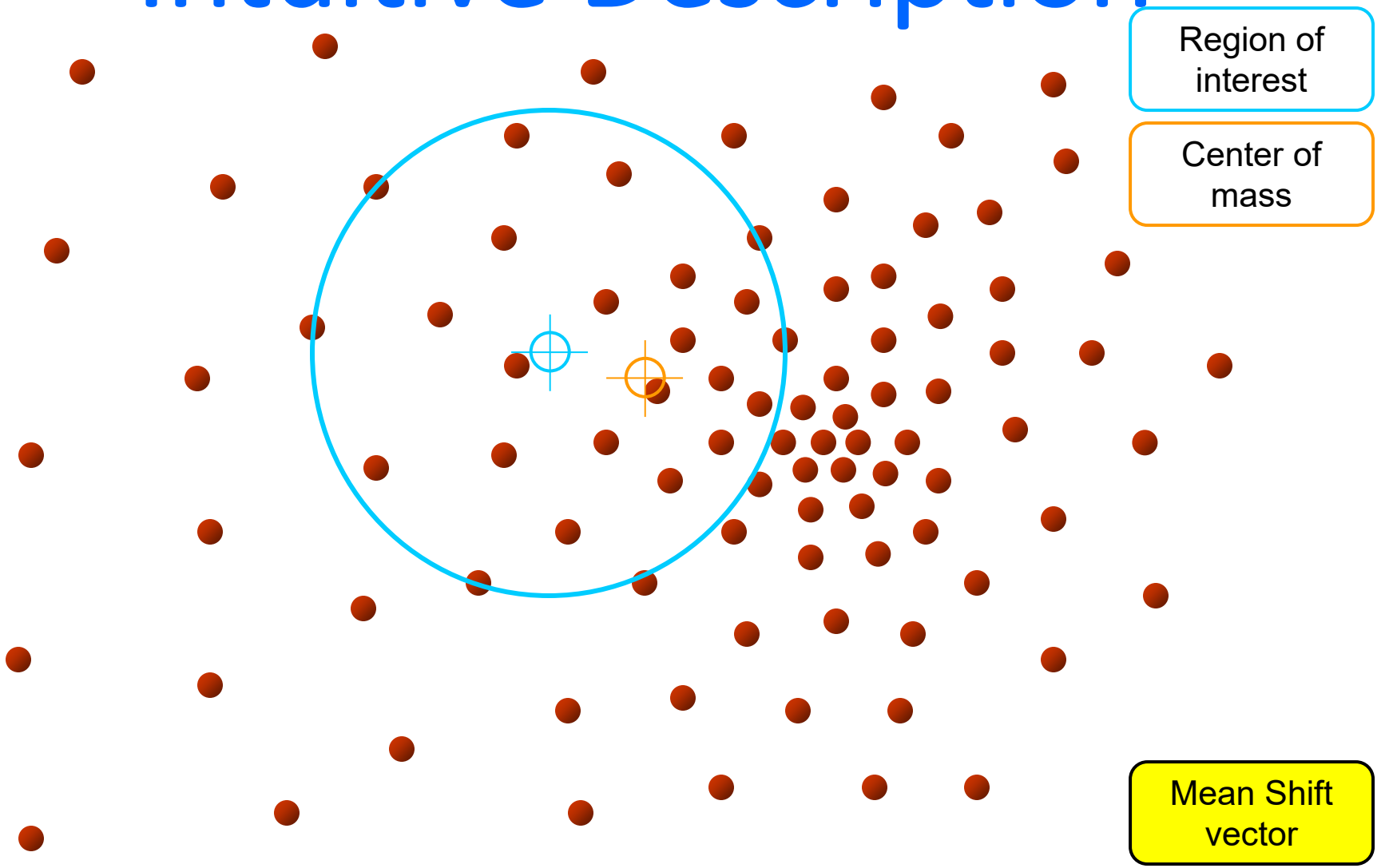
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

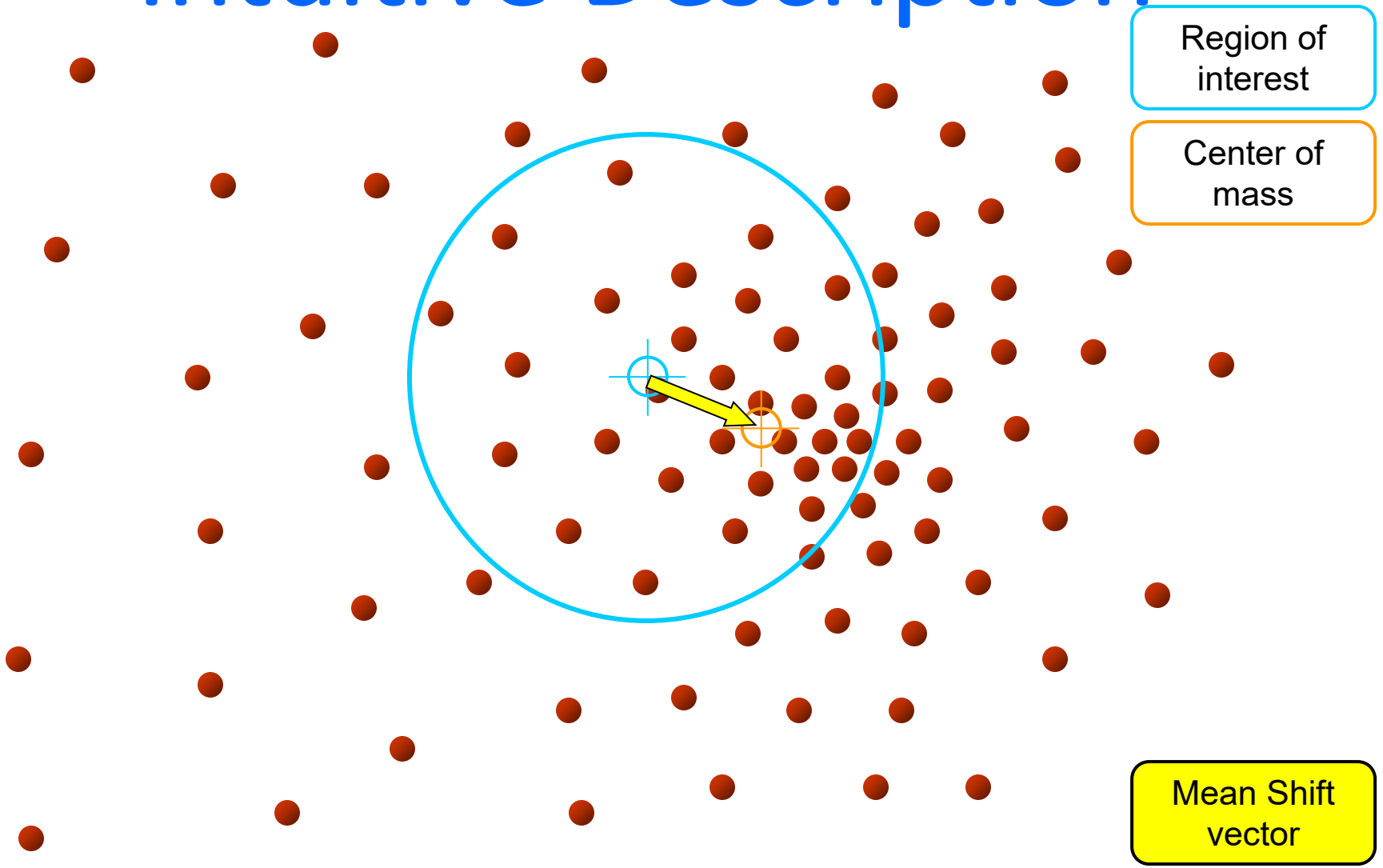
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

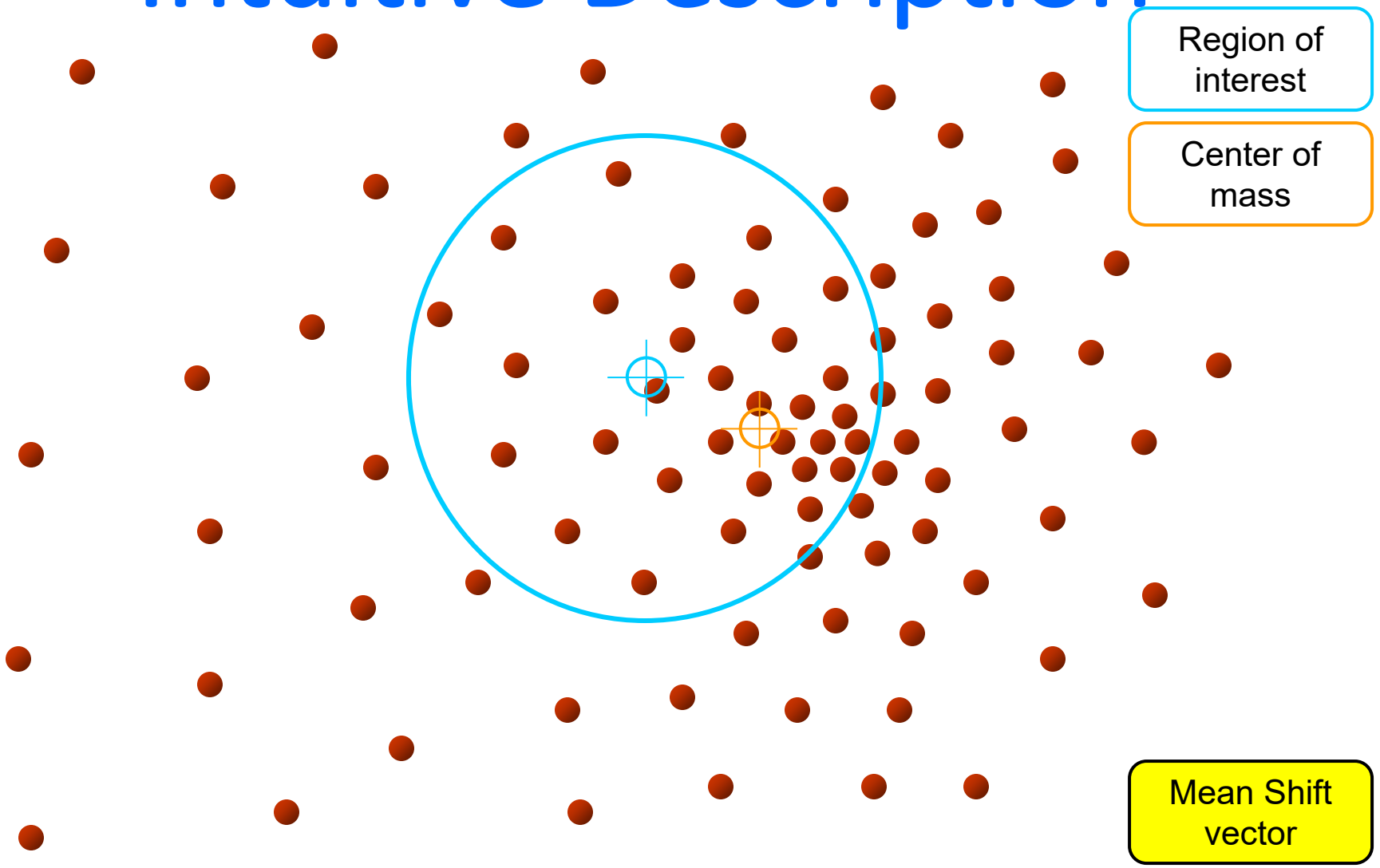
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

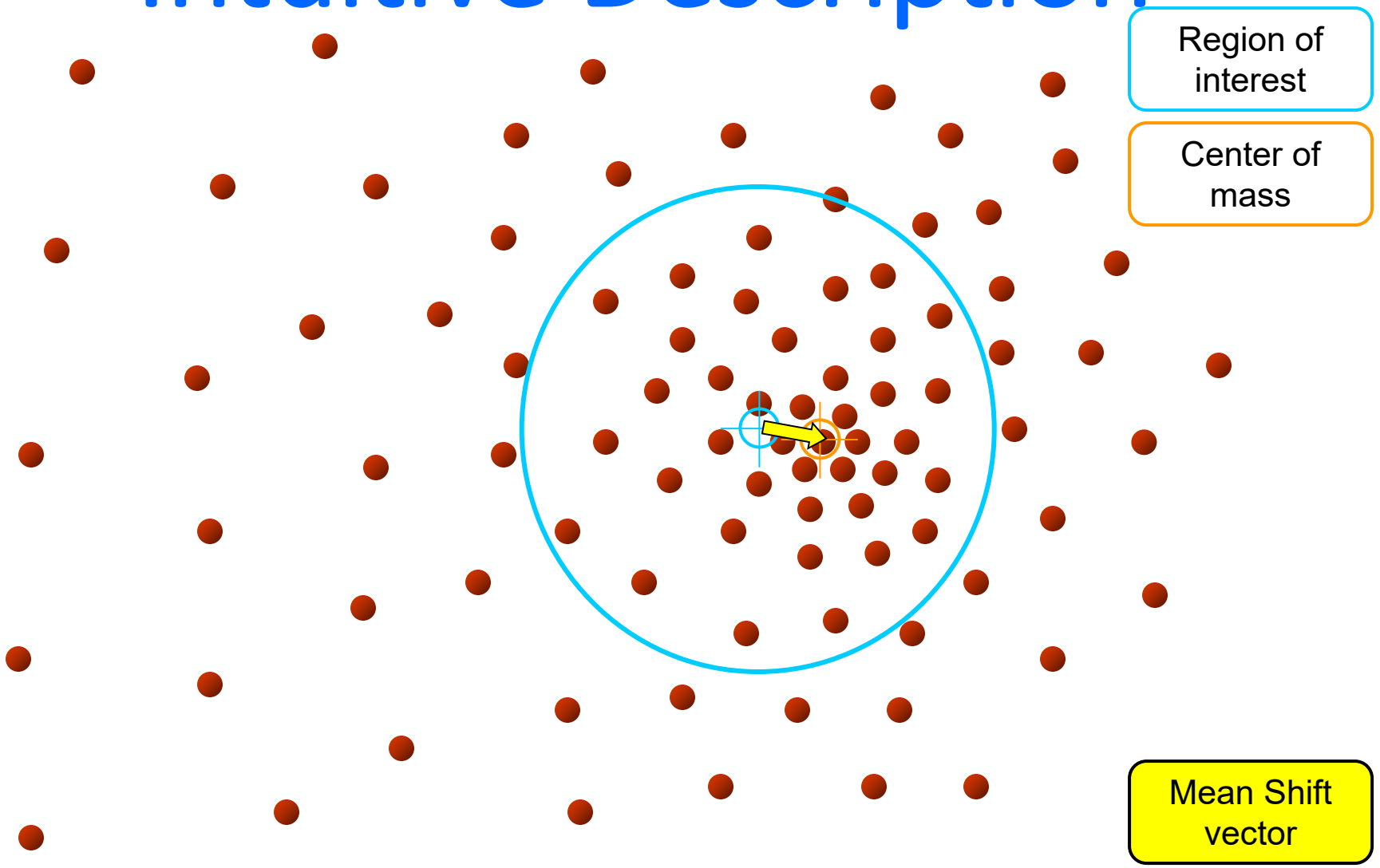
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

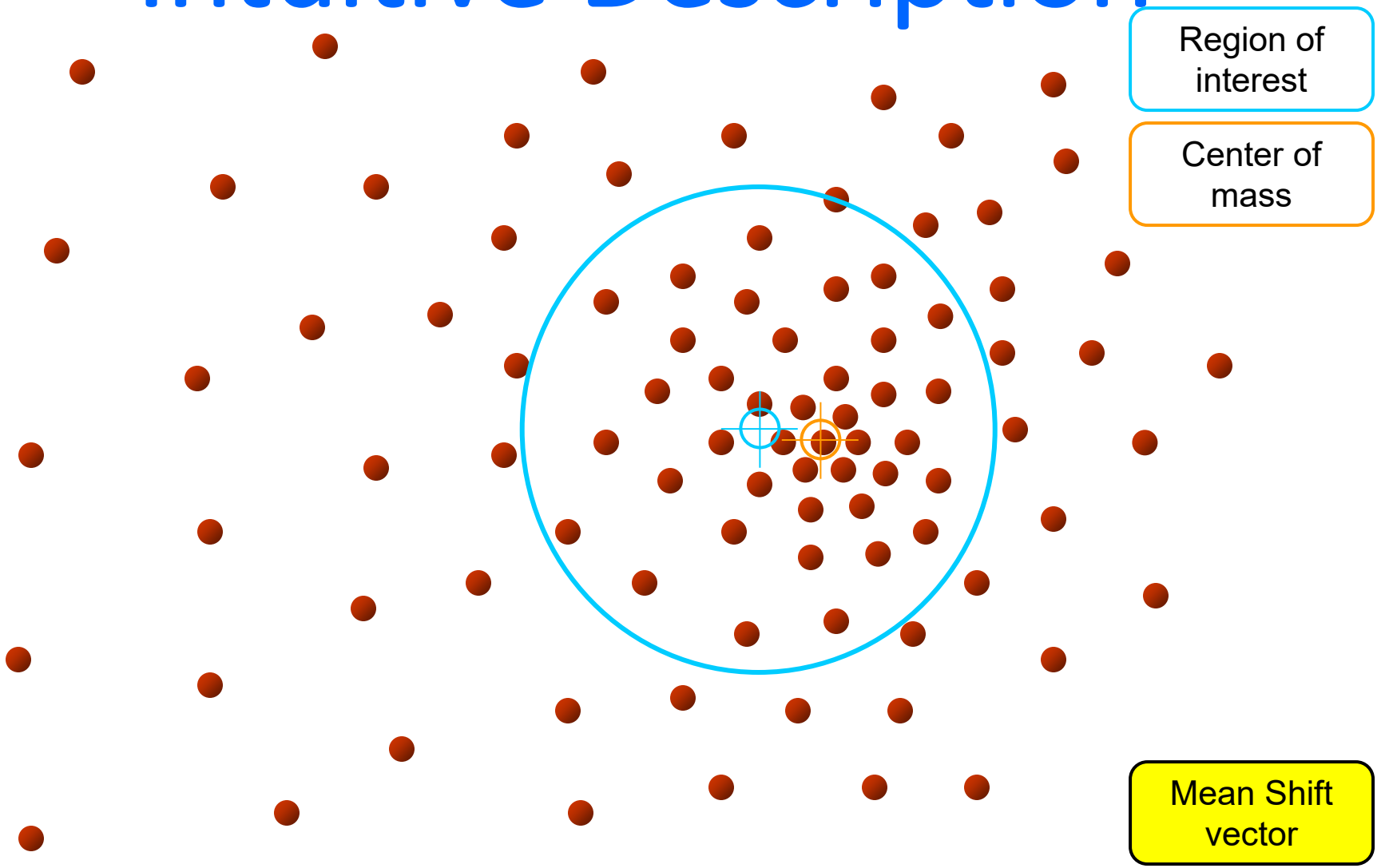
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

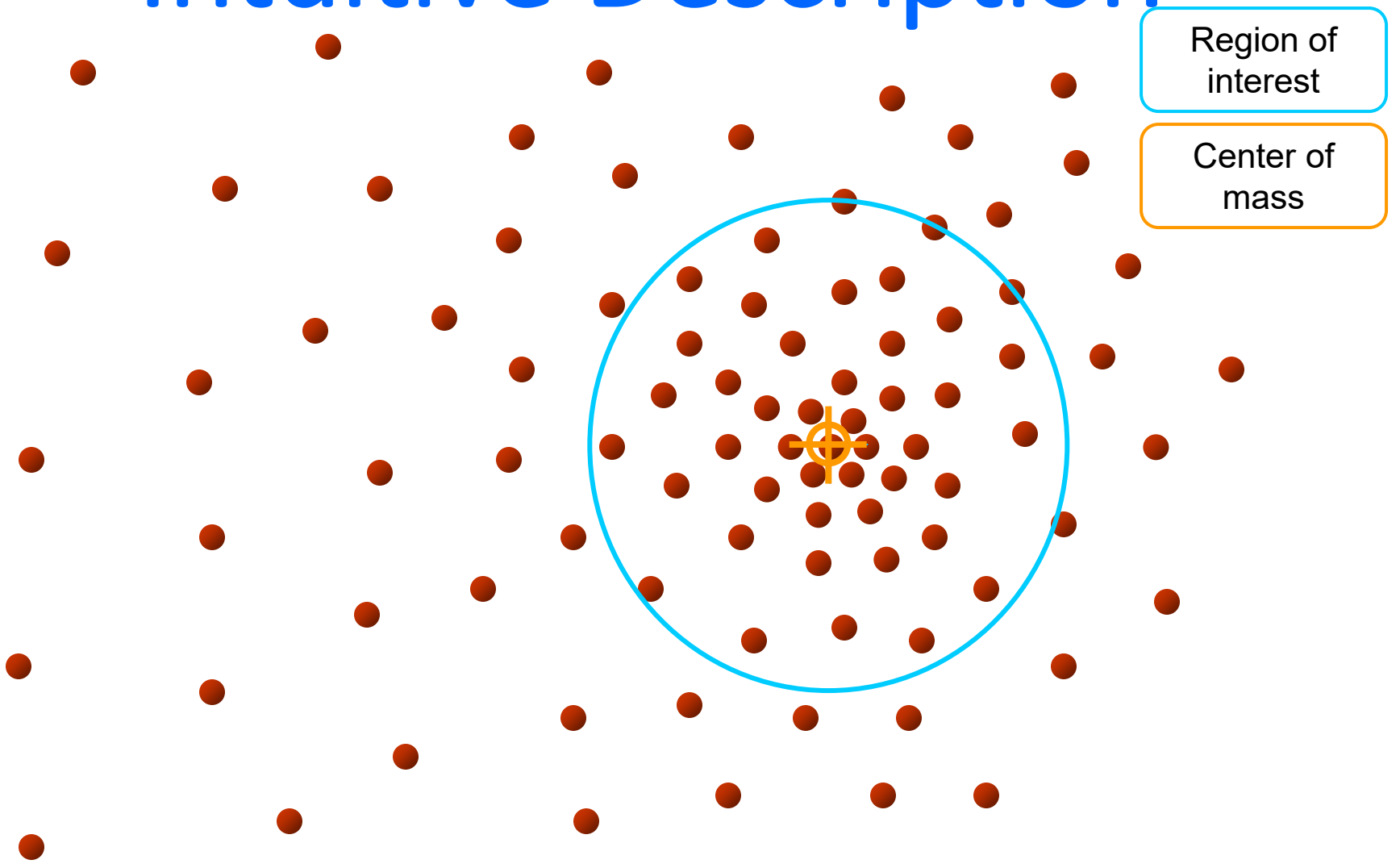
Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

Intuitive Description



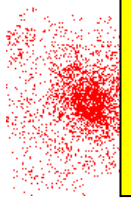
Objective : Find the densest region

Distribution of identical billiard balls

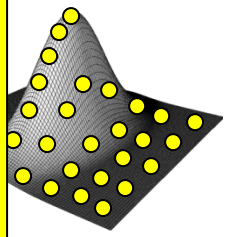
What is Mean Shift ?

A tool for:
Finding modes in a set of data samples, manifesting an underlying probability density function (PDF) in R^N

PDF in feature space
- **Color space**
- **Scale space**
- **Actually any feature space you can conceive**
- **…**

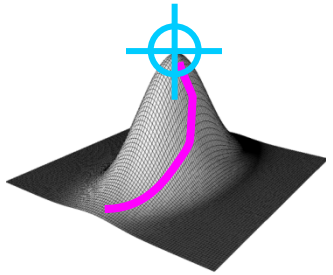


Data



DF Representation

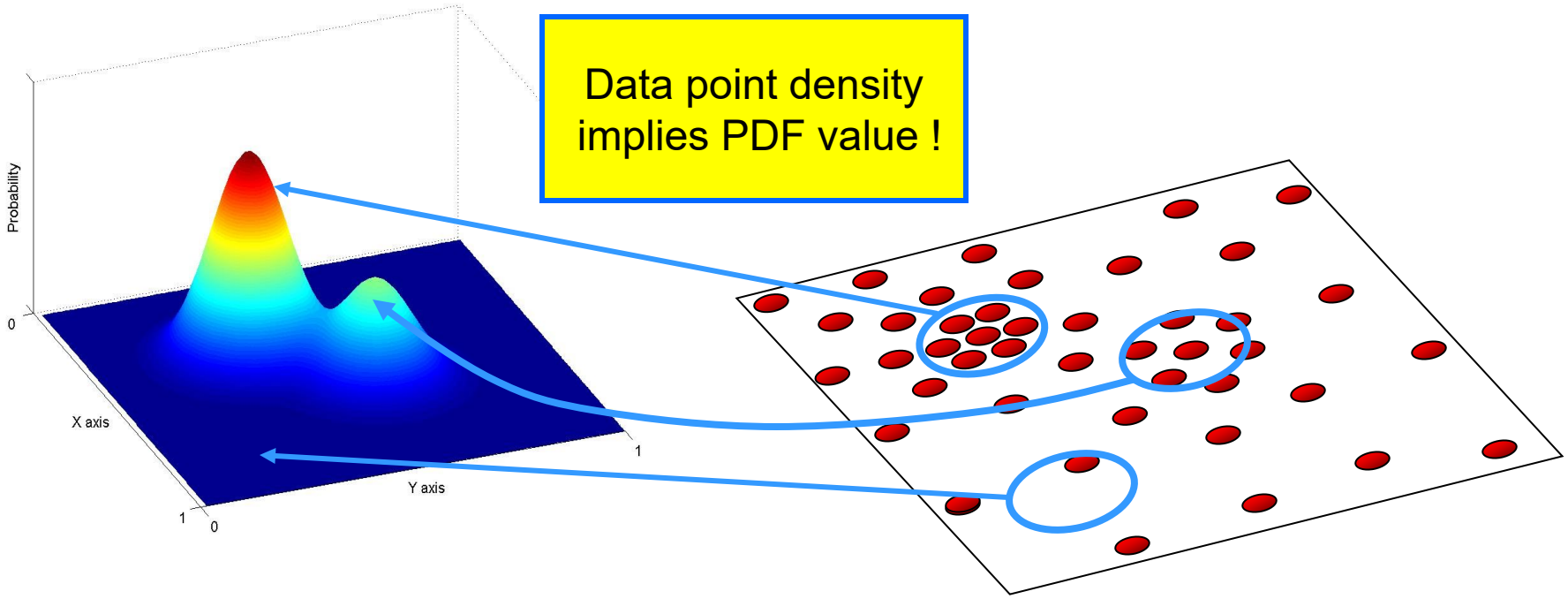
Non-parametric
Density **GRADIENT** Estimation
(Mean Shift)



PDF Analysis

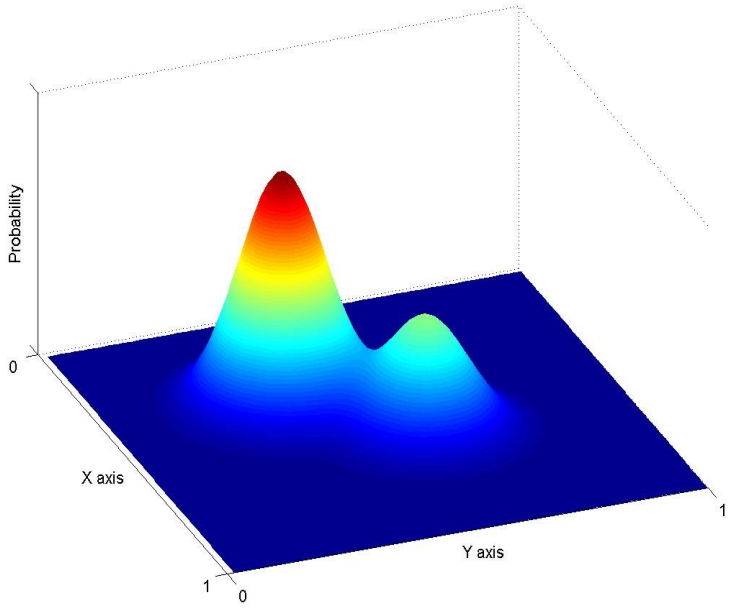
Non-Parametric Density Estimation

<u>Assumption</u> : The data points are sampled from an underlying PDF

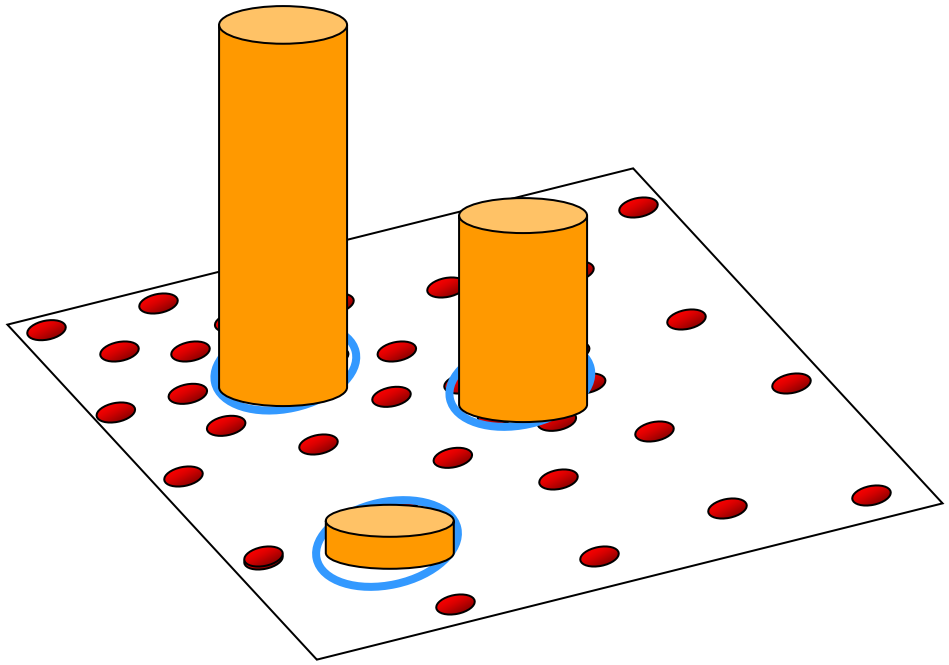


Assumed Underlying PDF **Real Data Samples**

Non-Parametric Density Estimation

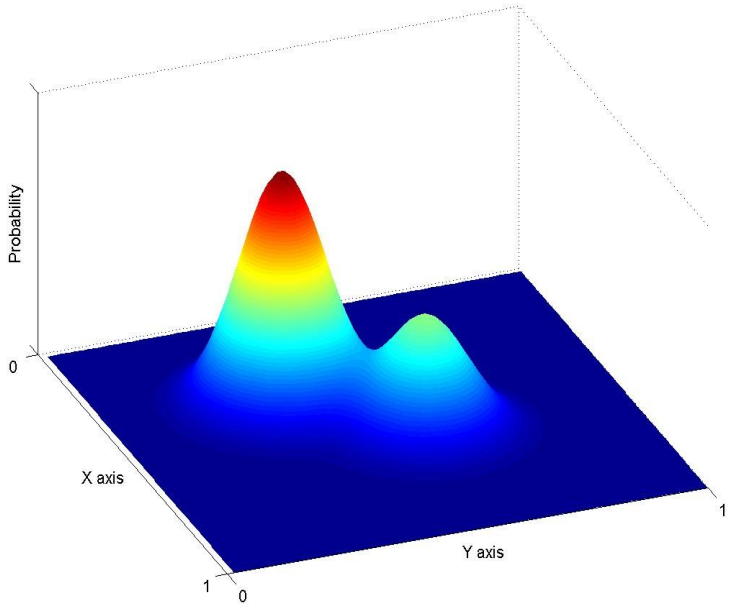


Assumed Underlying PDF

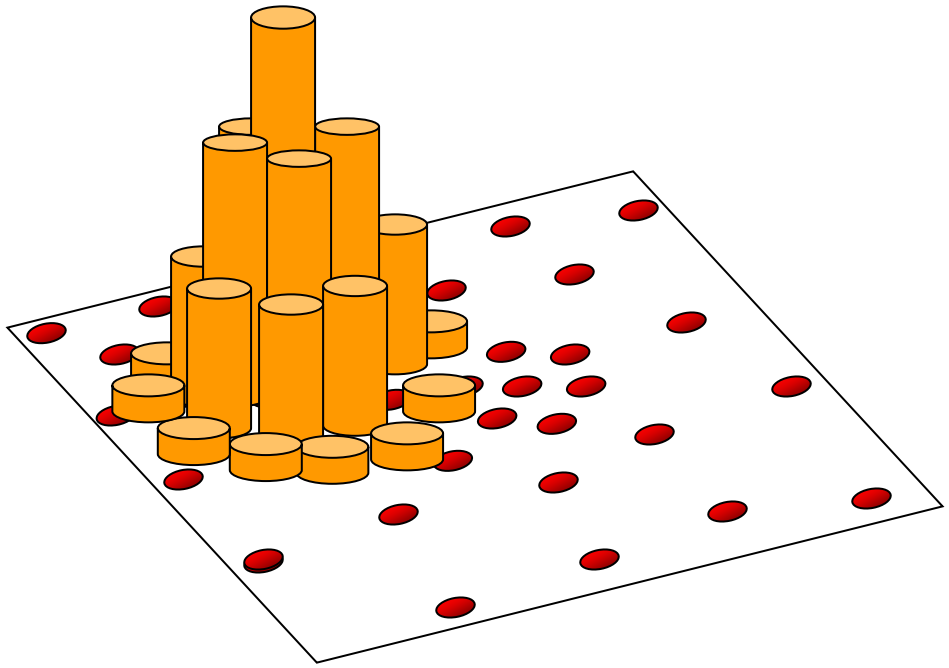


Real Data Samples

Non-Parametric Density Estimation



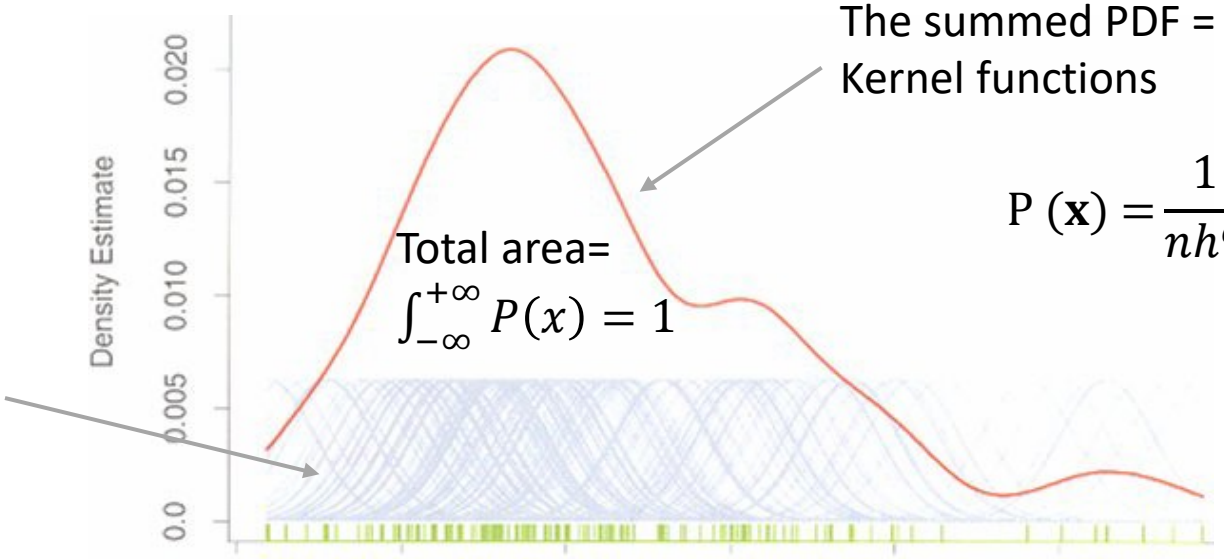
Assumed Underlying PDF



Real Data Samples

Parzen Estimation Aka Kernel Density Estimation

- **Parzen windows**: Approximate probability density by estimating local density of points (same idea as a histogram)
  - – Convolve points with window/kernel function (e.g., Gaussian) using scale parameter (e.g., sigma)



The summed PDF = Sum of many small Gaussian Kernel functions

Total area= $\int_{-\infty}^{+\infty} P(x) = 1$

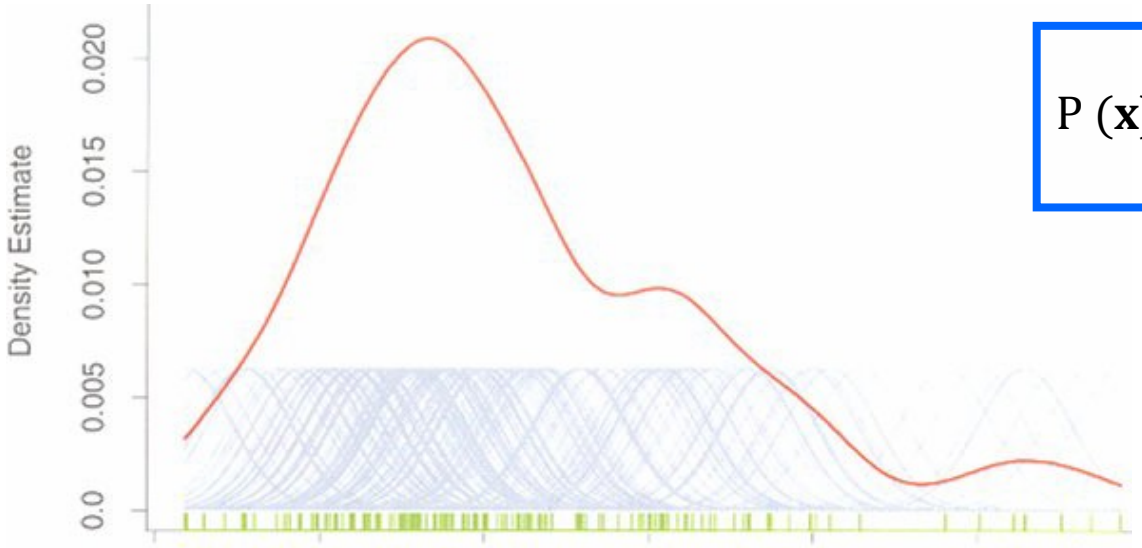
$$\mathrm{P}\,(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Kernel function at each sample points

Kernel Function $K\left(\frac{x-x_i}{h}\right)$ represents the 'distance' between two points(x, x_i)

Parzen Estimation Aka Kernel Density Estimation

- **Parzen windows**: Approximate probability density by estimating local density of points (same idea as a histogram)
  - – Convolve points with window/kernel function (e.g., Gaussian) using scale parameter (e.g., sigma)



$$\mathrm{P}(\mathbf{x}) = \sum_{\mathrm{i}} c_i \cdot e^{-\frac{(\mathbf{x}-\boldsymbol{\mu}_i)^2}{2{\sigma_i}^2}}$$

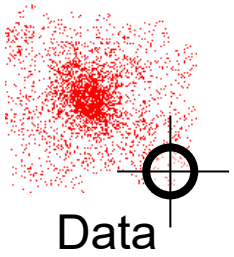
Sum of Gaussian contribution aka 'Kernel'
Different types of Kernels !

Kernel Density Estimation

Various Kernels

$$\mathrm{P}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

A function of some finite number of data points $x_1 \ldots x_n$



n = *number of samples*
h = *bandwidth (window radius)*
d = *dimension*
x = *target position*
x_i = *samples*

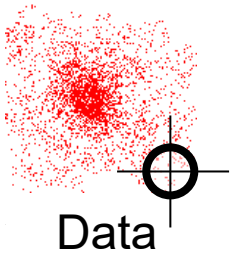
Kernel Function $K\left(\frac{x-x_i}{h}\right)$ represents the 'distance' between two points(x, x_i)

Kernel Density Estimation

Various Kernels

$$\mathrm{P}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

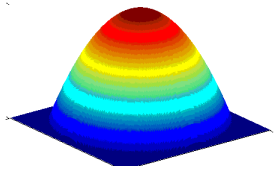
A function of some finite number of data points $x_1 \ldots x_n$



Examples:

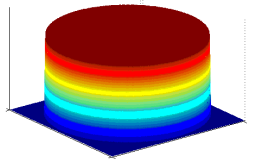
- Epanechnikov Kernel

$$K_E(\mathbf{x}) = \begin{cases} c\left(1 - \|\mathbf{x}\|^2\right) & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



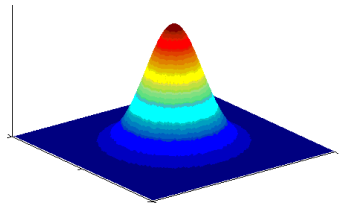
- Uniform Kernel

$$K_U(\mathbf{x}) = \begin{cases} c & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



- Normal Kernel (Gaussian)

$$K_N(\mathbf{x}) = c \cdot \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$

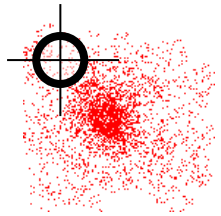


Relating KDE to Mean-Shift

- Given:

- Finite number of data points $x_1 \ldots x_n$

  A Kernel Function $K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$



Data

- Goal:

- *Find the mean sample point* $\mathbf{x}$
- *Find the peak of the Probability Density Estimation*

How is KDE related to mean shift?

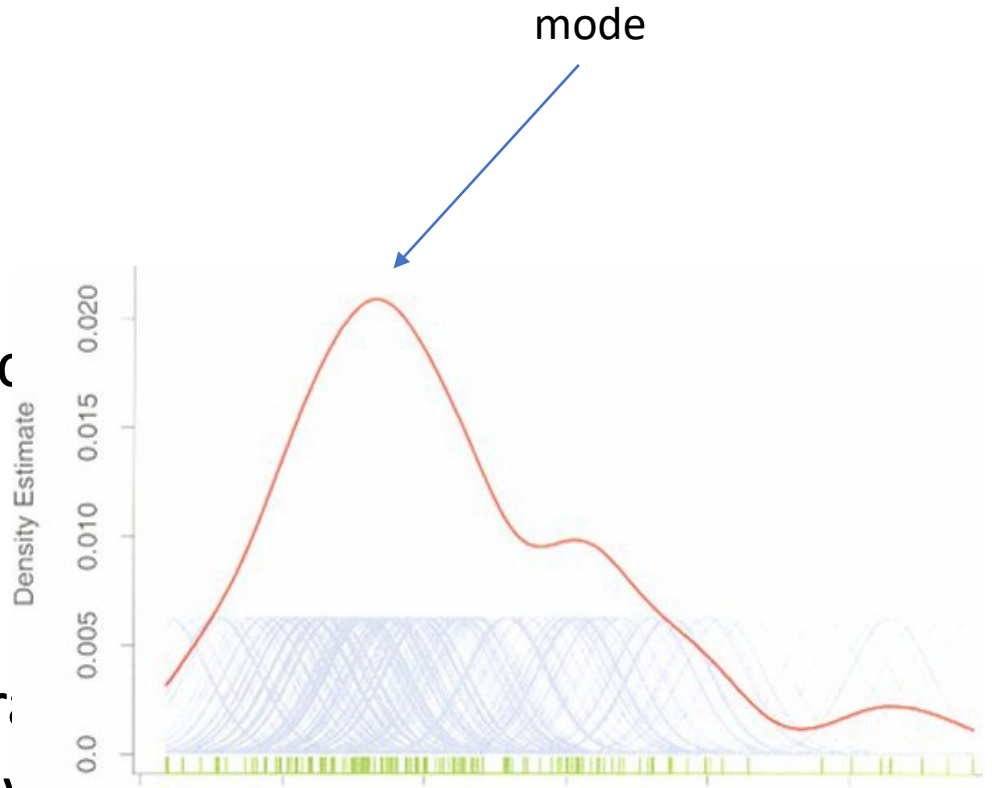
- Recall KDE can be expressed as:
-

$$P(\mathbf{x}) = \frac{1}{nh^d} c \sum_{i=1}^{n} k\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)$$

- Gradient of PDE is related to the mean shift vec

$$\nabla P(\mathbf{x}) \propto \mathbf{v}(\mathbf{x})$$

- The mean shift vector in the direction of the gr
- Mean-shift algorithm is maximizing the objecti



How is KDE related to mean shift?

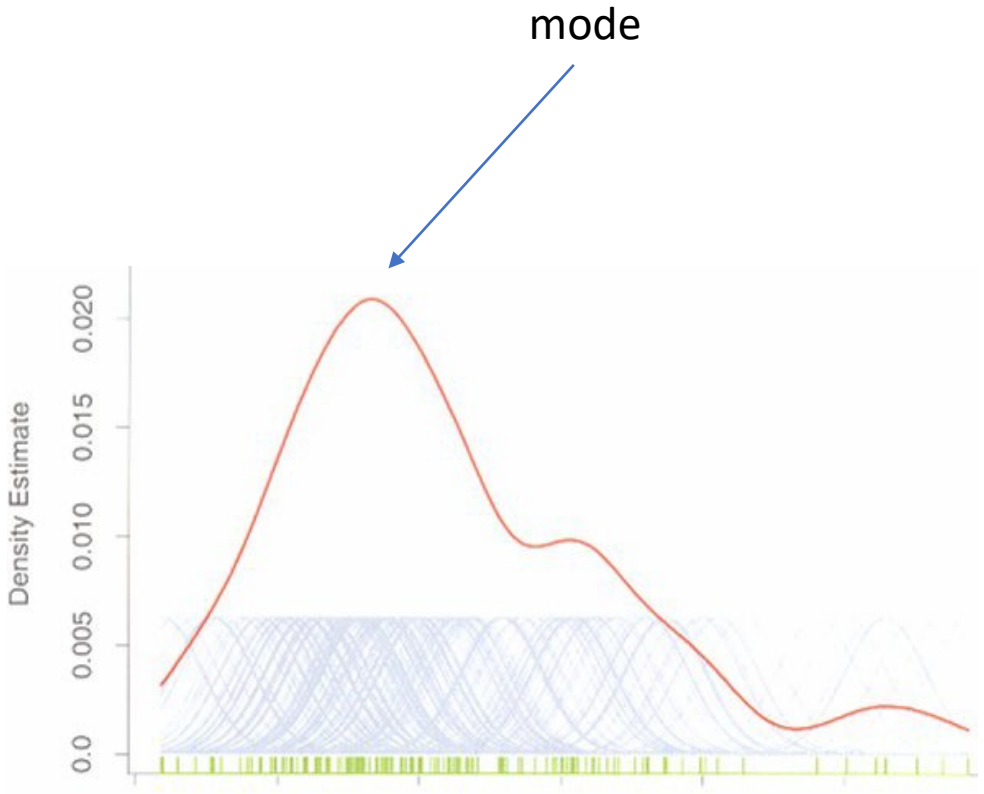
- In mean-shift we are trying to optimize

$$\arg\max_{\mathbf{x}} P(\mathbf{x})$$

$$\arg\max_{\mathbf{x}} = \frac{1}{nh^d} c \sum_{i=1}^{n} k\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)$$

- How to do this?

Using gradient descent!



$\nabla P(\mathbf{x}) \propto mean\ shift\ vector$

- The equation becomes:

$$\nabla P(\mathbf{x}) = \frac{1}{n} 2c \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\left(\frac{\mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}\right)$$

- Remember in mean shift, we have

$$\mathbf{v}(\mathbf{x}) = \mathbf{m}(\mathbf{x}) - \mathbf{x}$$

$$\mathbf{v}(\mathbf{x}) = \frac{\nabla P(\mathbf{x})}{\frac{1}{n} 2c \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}$$

Summary - Mean-Shift Algorithm

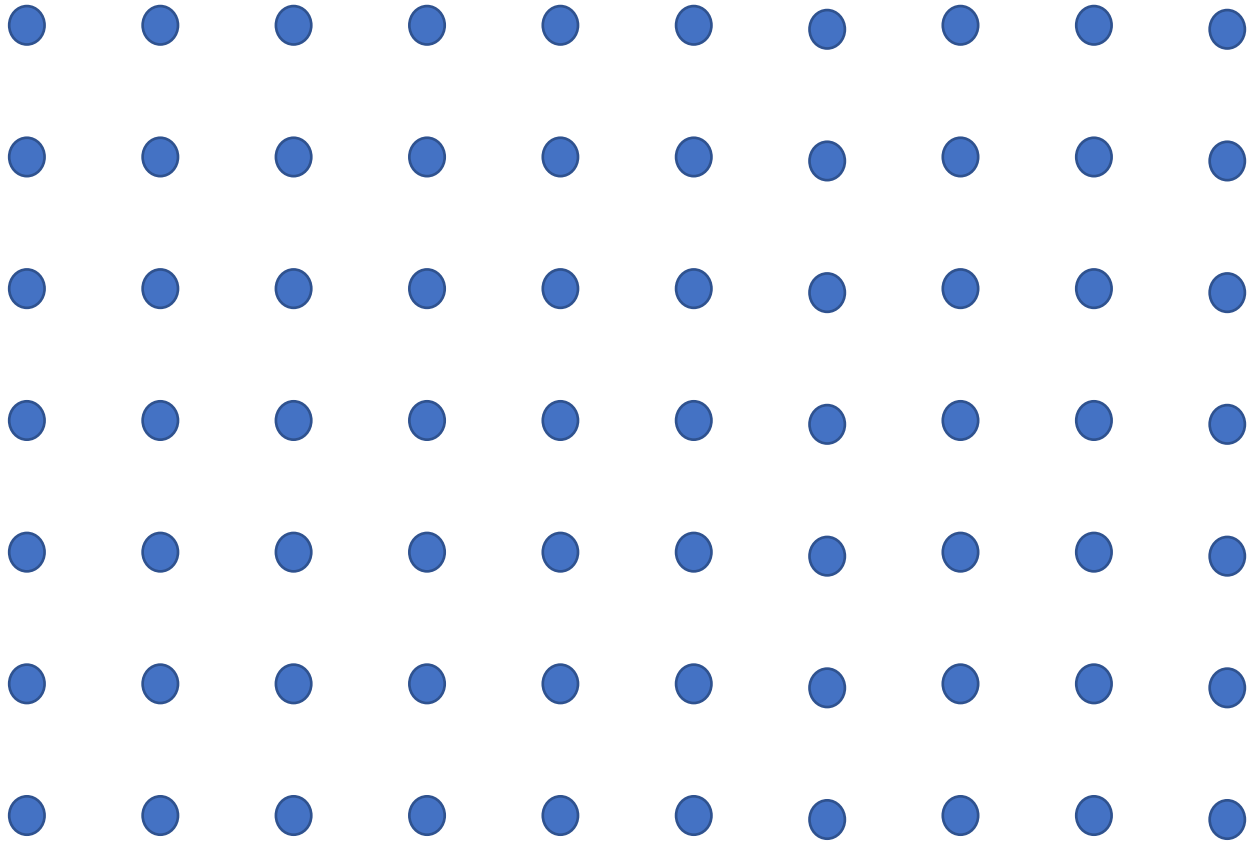
- Steps for mean-shift

- 1. Initialize $\mathbf{x}$
- 2. Compute mean $\mathbf{m}(\mathbf{x}) = \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i K(\frac{\mathbf{x}-\mathbf{x}_i}{h})}{\sum_{i=1}^{n} K(\frac{\mathbf{x}-\mathbf{x}_i}{h})}\right]$
- 3. Compute the shift $\mathbf{v}(\mathbf{x}) = \mathrm{m}(\mathbf{x}) - \mathbf{x}$
- 4. update $\mathbf{x} = \mathbf{x} + \mathbf{v}(\mathbf{x})$
- 5. if $\mathbf{v}(\mathbf{x}) > \varepsilon$ repeat 2~4

$$\mathbf{v}(\mathbf{x}) = \frac{\nabla P(\mathbf{x})}{\frac{1}{n} 2c \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}$$

Mean Shift with Images

- Image is represented as an array
- Density is the same everywhere
- How to apply Mean-Shift?



Mean-Shift with Images

- Let pixels form a uniform grid of data points, each with a ***weight*** (pixel value) proportional to the "likelihood" that the pixel is on the object we want to track.

Consider a set of points: $\{\mathbf{x}_i\}_{i=1,\ldots n}$

Corresponding weight: $w(\mathbf{x}_i)$

Sample mean: $m(\mathbf{x}) = \frac{\sum_i K(\mathbf{x}-\mathbf{x}_i) w(\mathbf{x}_i) \mathbf{x}_i}{\sum_i K(\mathbf{x}-\mathbf{x}_i) w(\mathbf{x}_i)}$

Mean Shift: $m(\mathbf{x}) - \mathbf{x}$

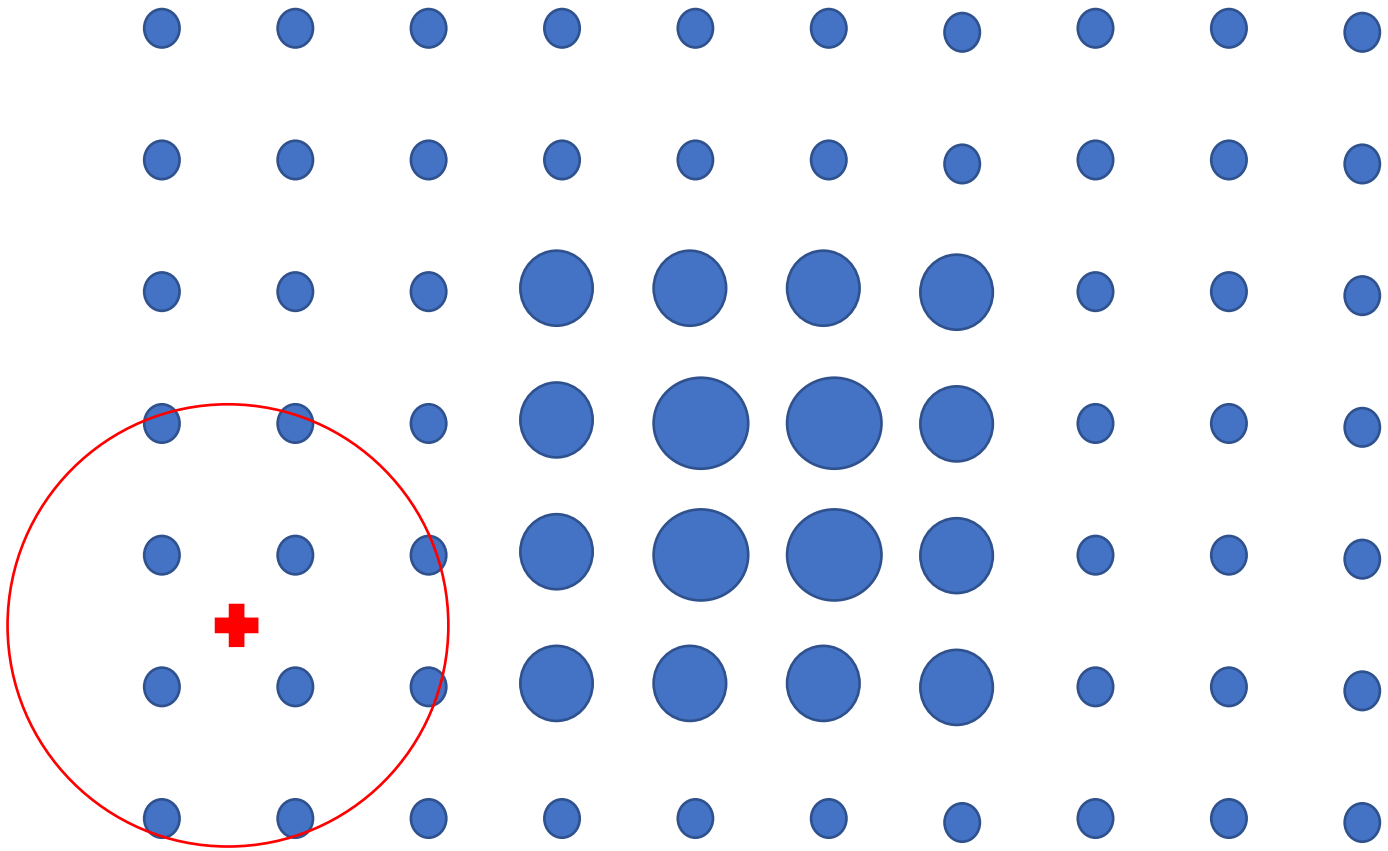
Mean-Shift Algorithm

- Steps for mean-shift

- 1. Initialize $\mathbf{x}$
- 2. Compute mean $\mathrm{m}(\mathbf{x}) = \frac{\sum_i K(\mathbf{x}-\mathbf{x}_i) w(\mathbf{x}_i) \mathbf{x}_i}{\sum_i K(\mathbf{x}-\mathbf{x}_i) w(\mathbf{x}_i)}$
- 3. Compute the shift $\mathbf{v}(\mathbf{x}) = \mathrm{m}(\mathbf{x}) - \mathbf{x}$
- 4. update $\mathbf{x} = \mathbf{x} + \mathbf{v}(\mathbf{x})$
- 5. if $\mathbf{v}(\mathbf{x}) > \varepsilon$ repeat 2~4

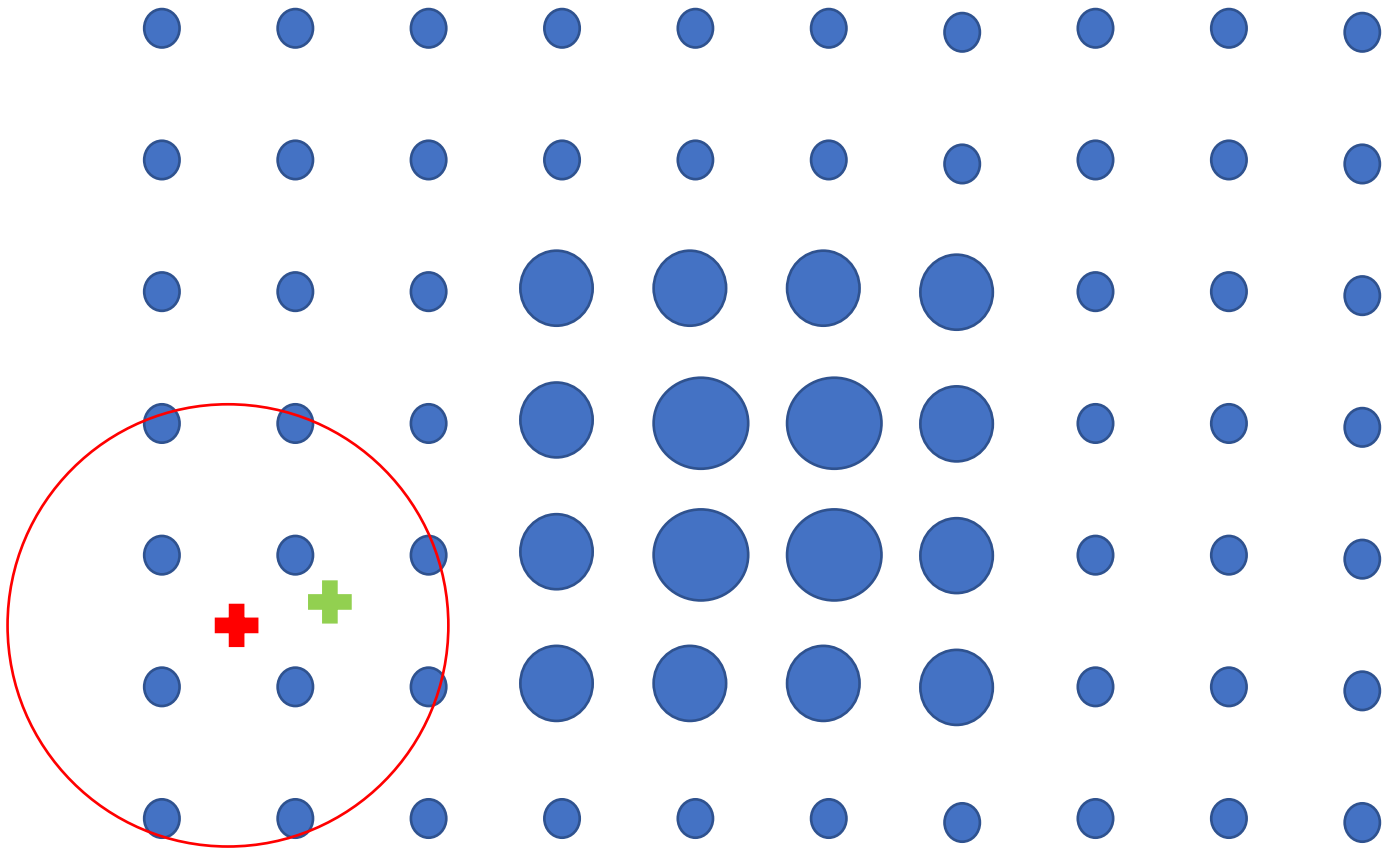
Mean Shift with Images

- Weight is assigned to each pixel points



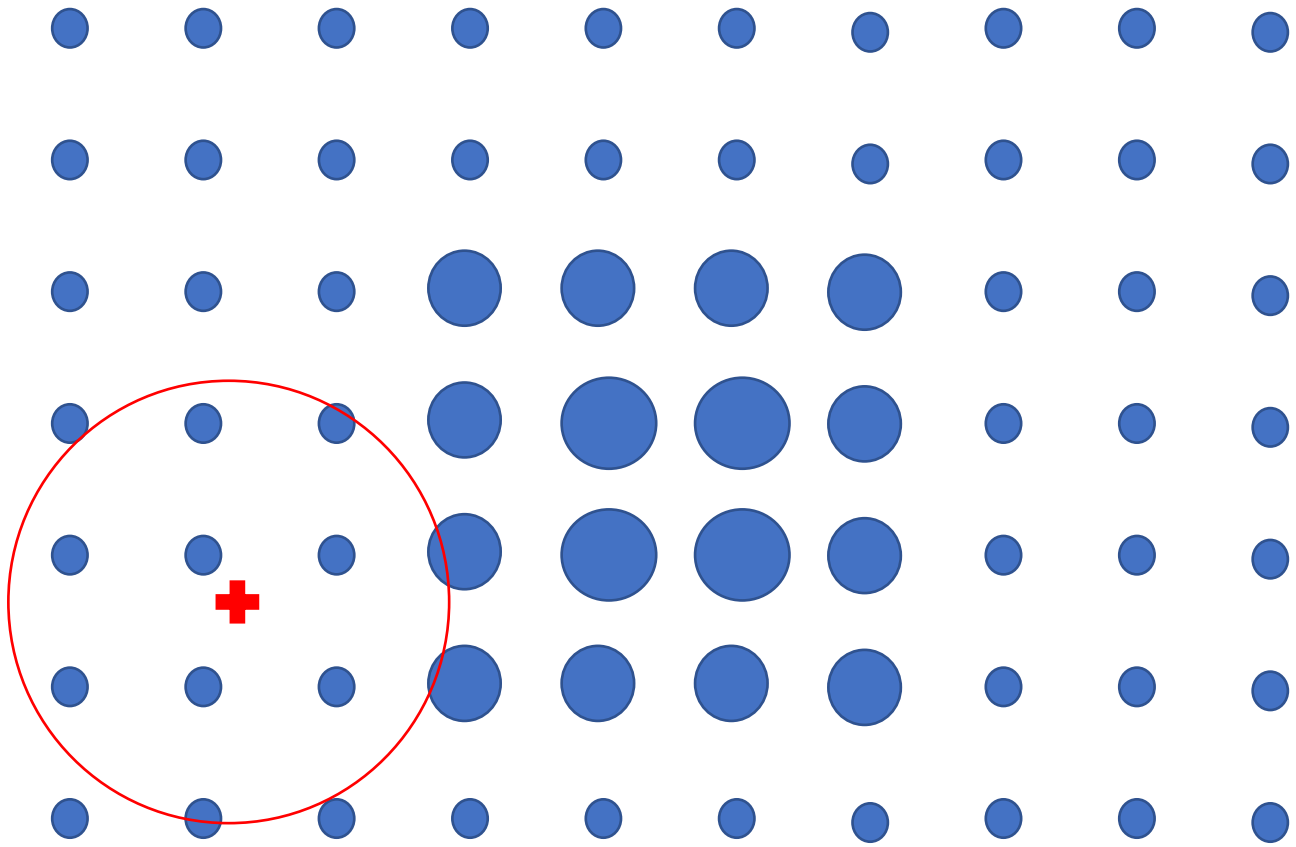
Mean Shift with Images

- Weight is assigned to each pixel points



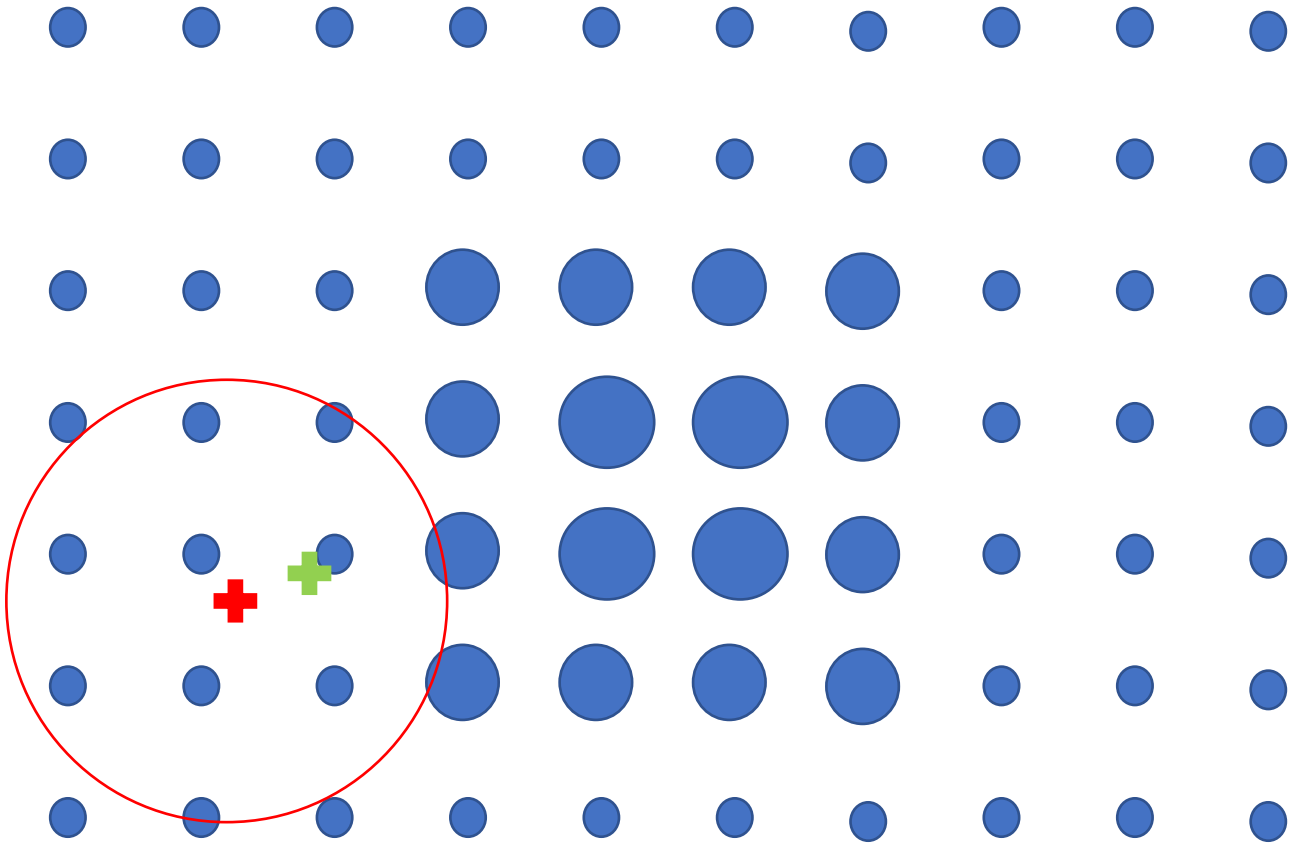
Mean Shift with Images

- Weight is assigned to each pixel points



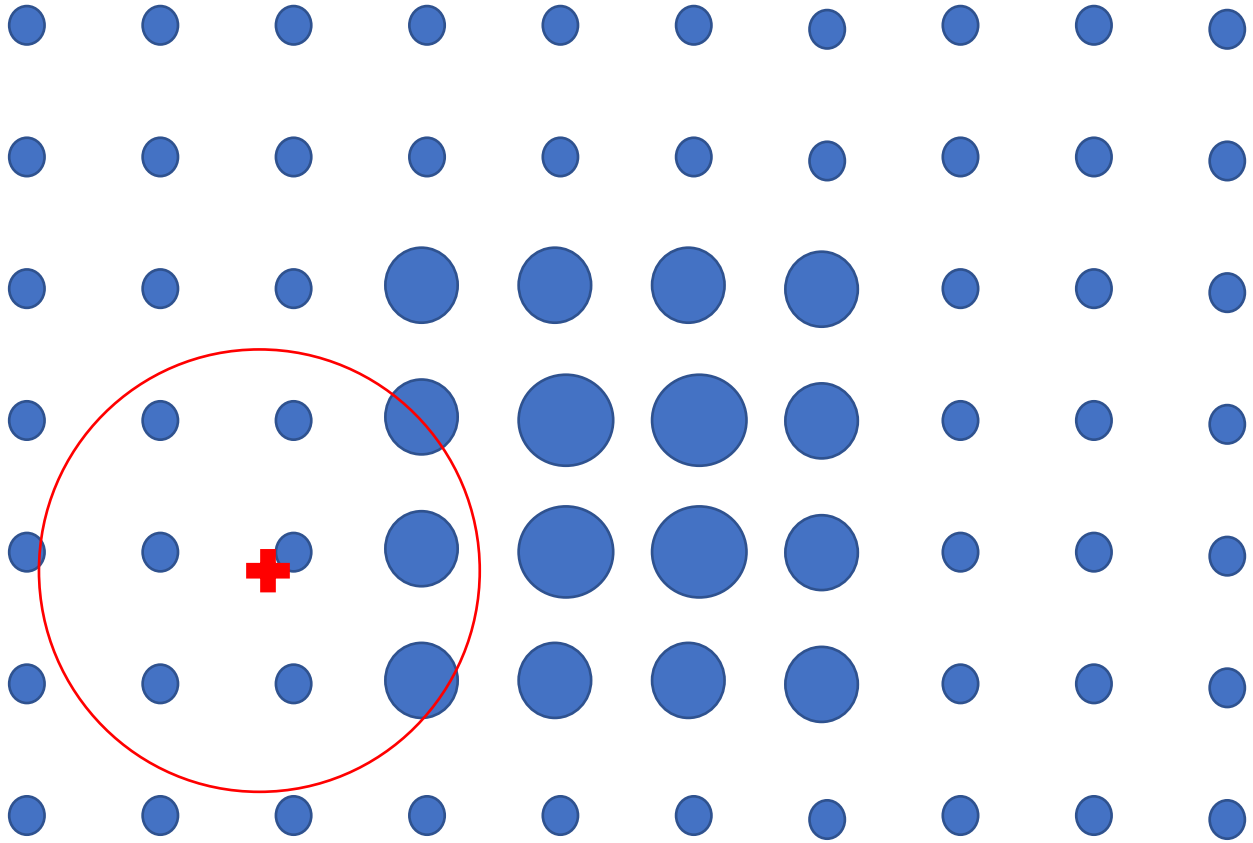
Mean Shift with Images

- Weight is assigned to each pixel points



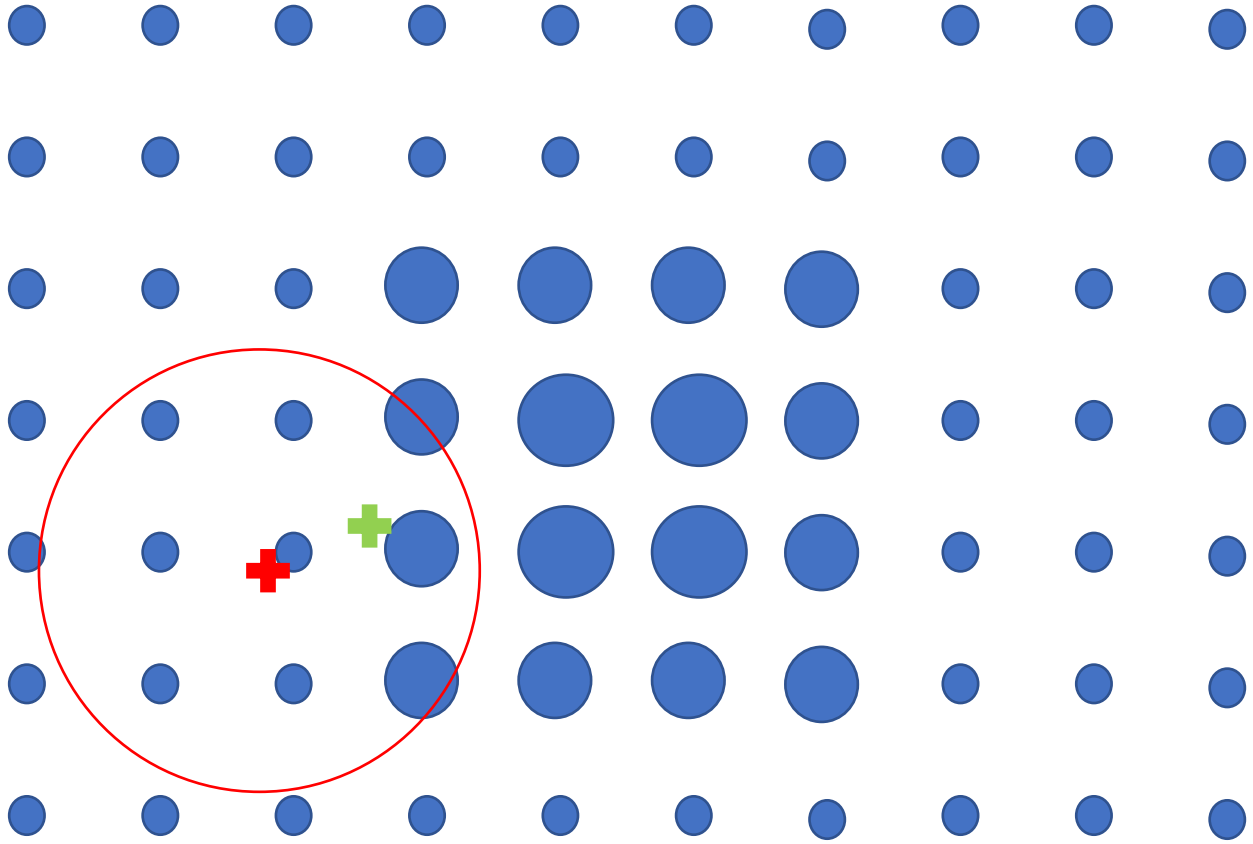
Mean Shift with Images

- Weight is assigned to each pixel points



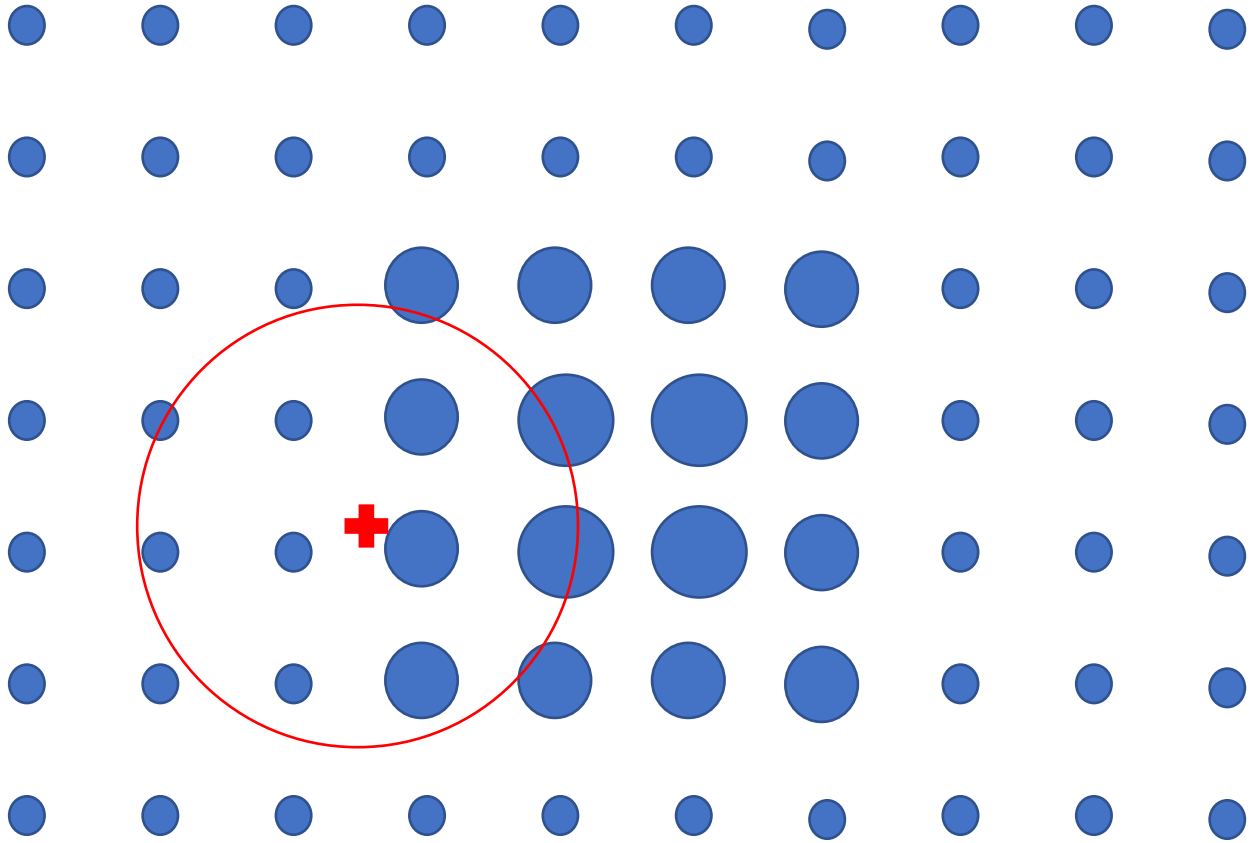
Mean Shift with Images

- Weight is assigned to each pixel points



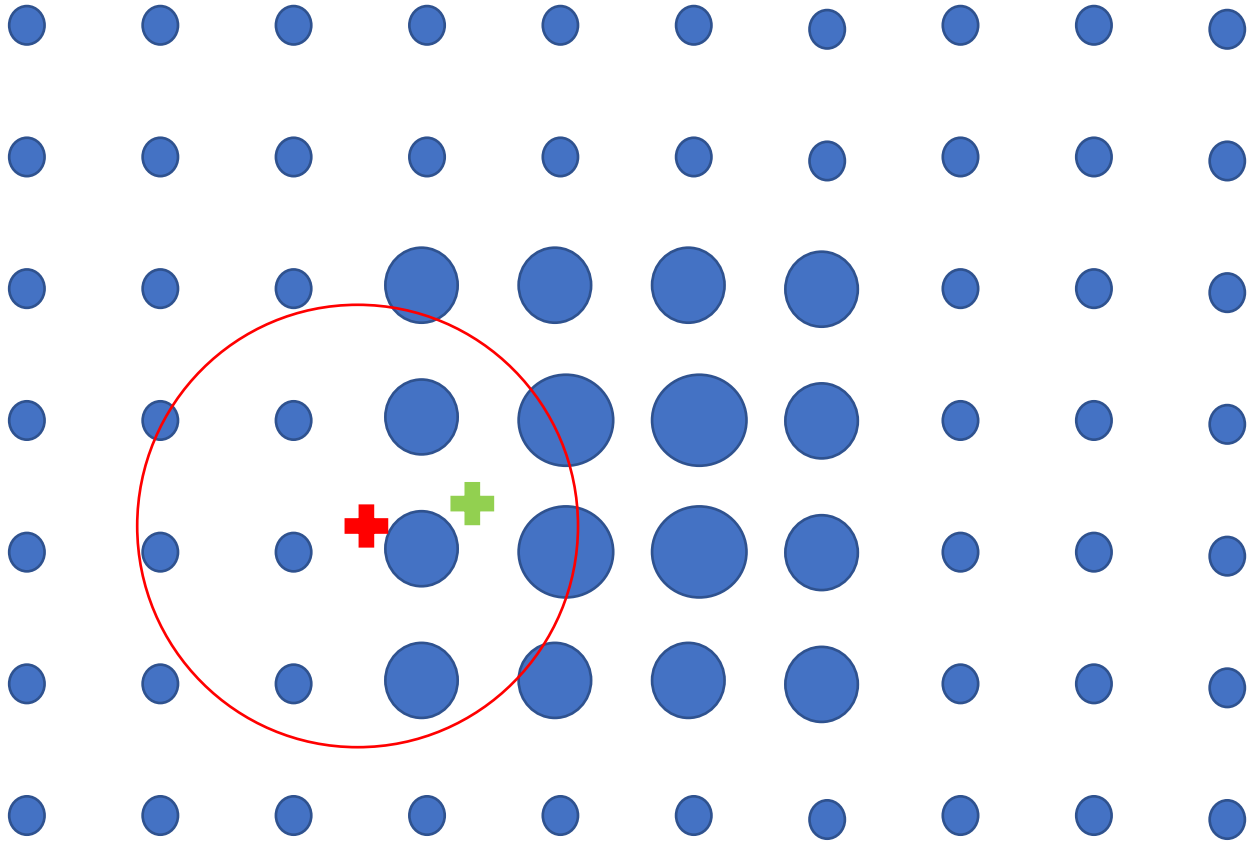
Mean Shift with Images

- Weight is assigned to each pixel points



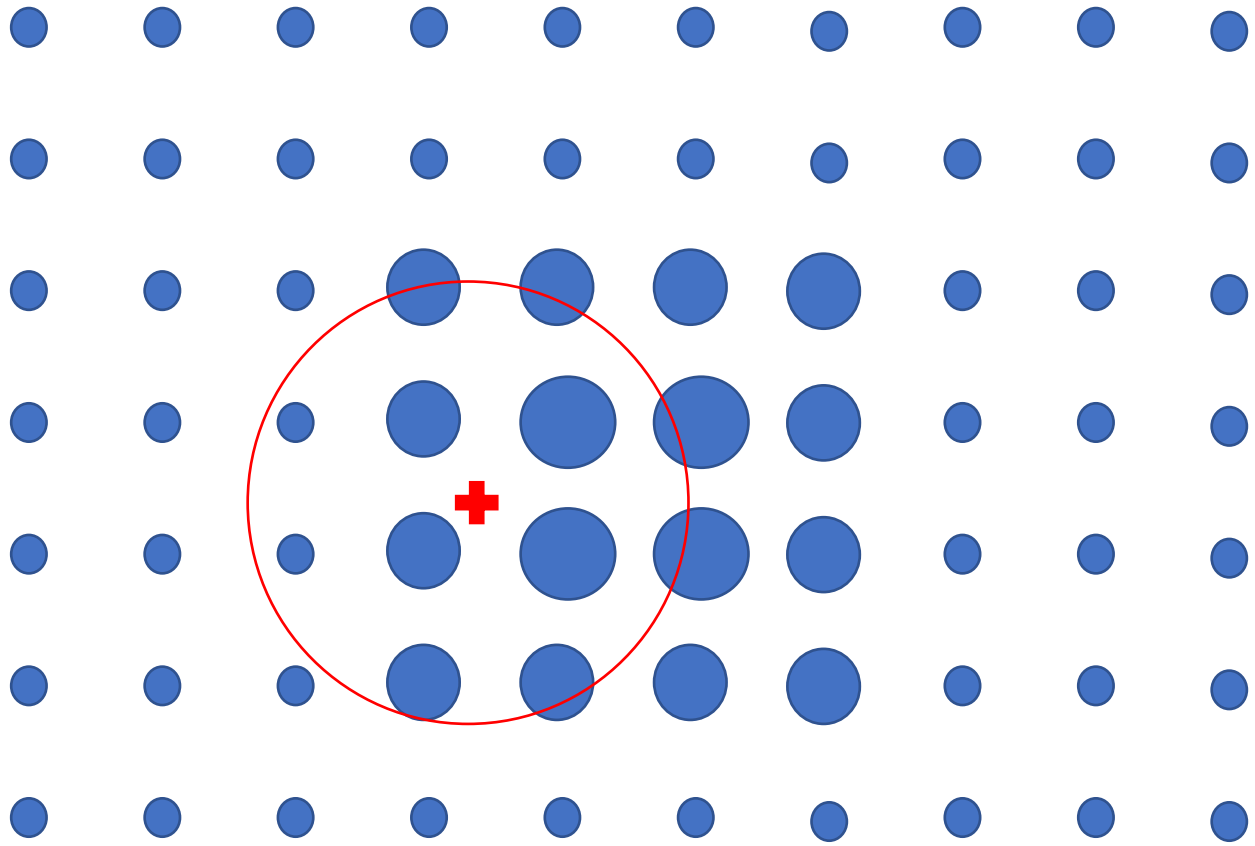
Mean Shift with Images

- Weight is assigned to each pixel points



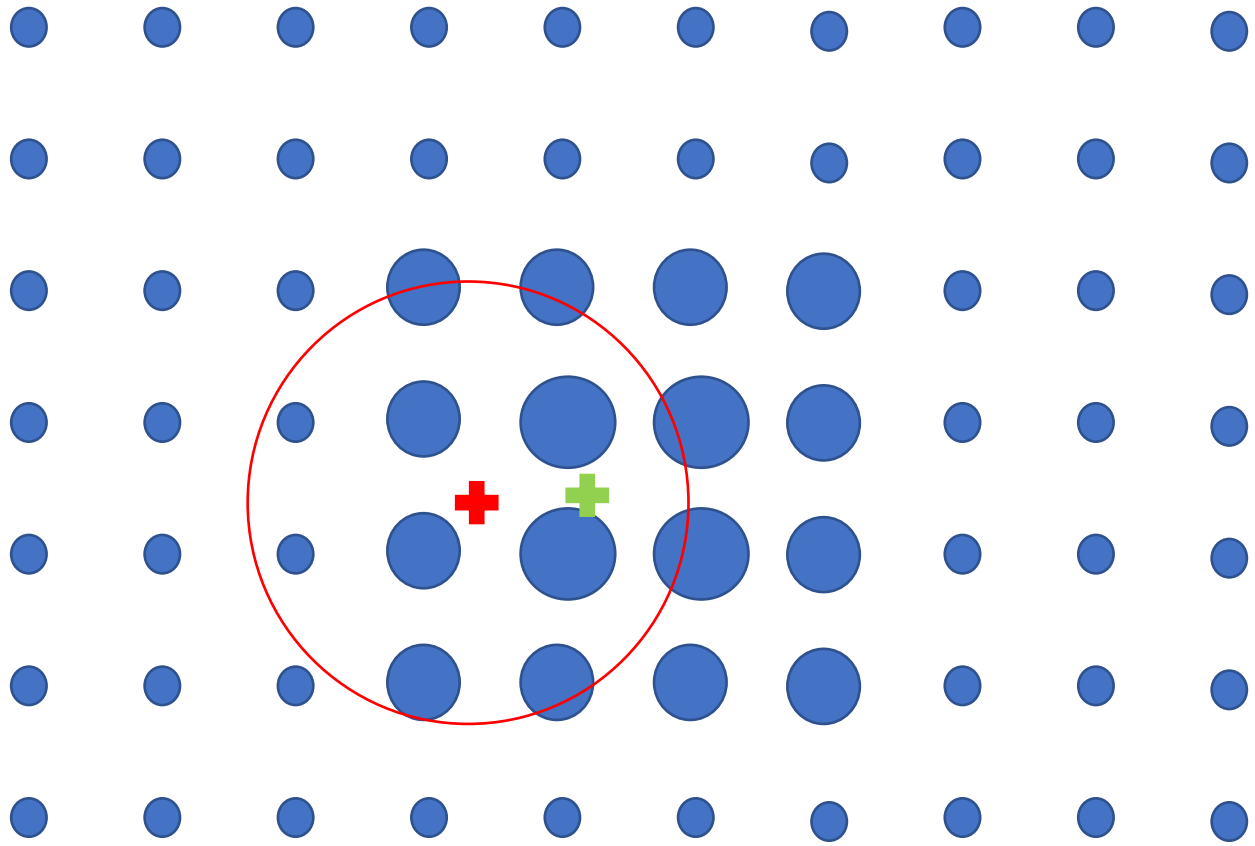
Mean Shift with Images

- Weight is assigned to each pixel points



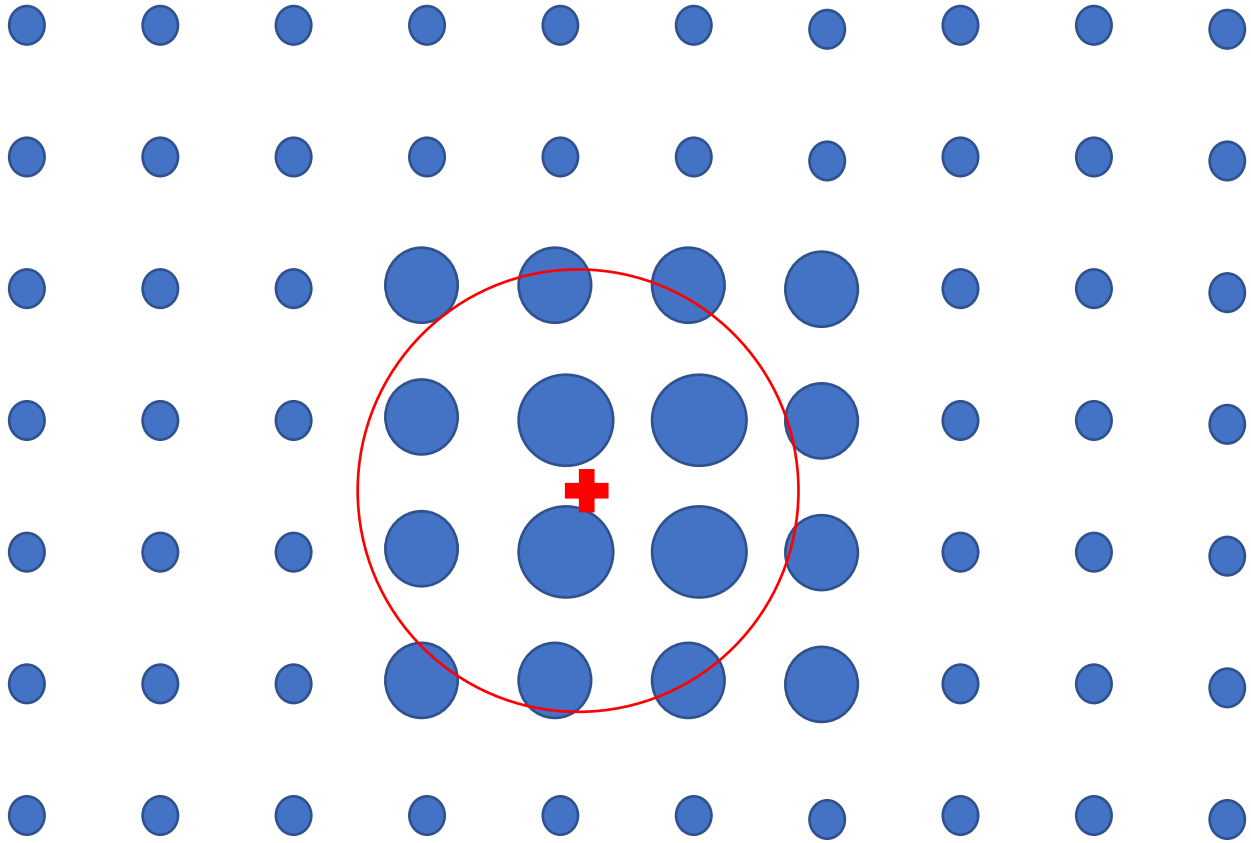
Mean Shift with Images

- Weight is assigned to each pixel points



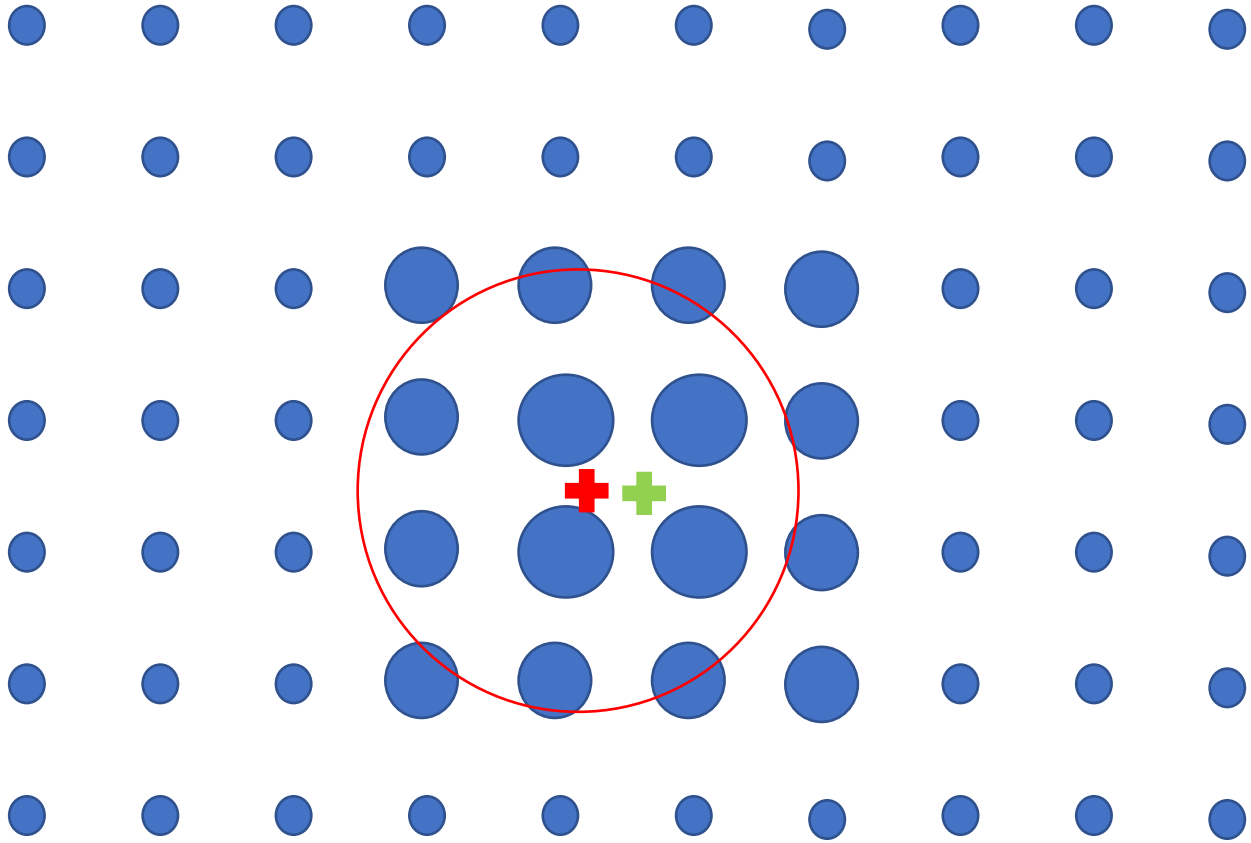
Mean Shift with Images

- Weight is assigned to each pixel points



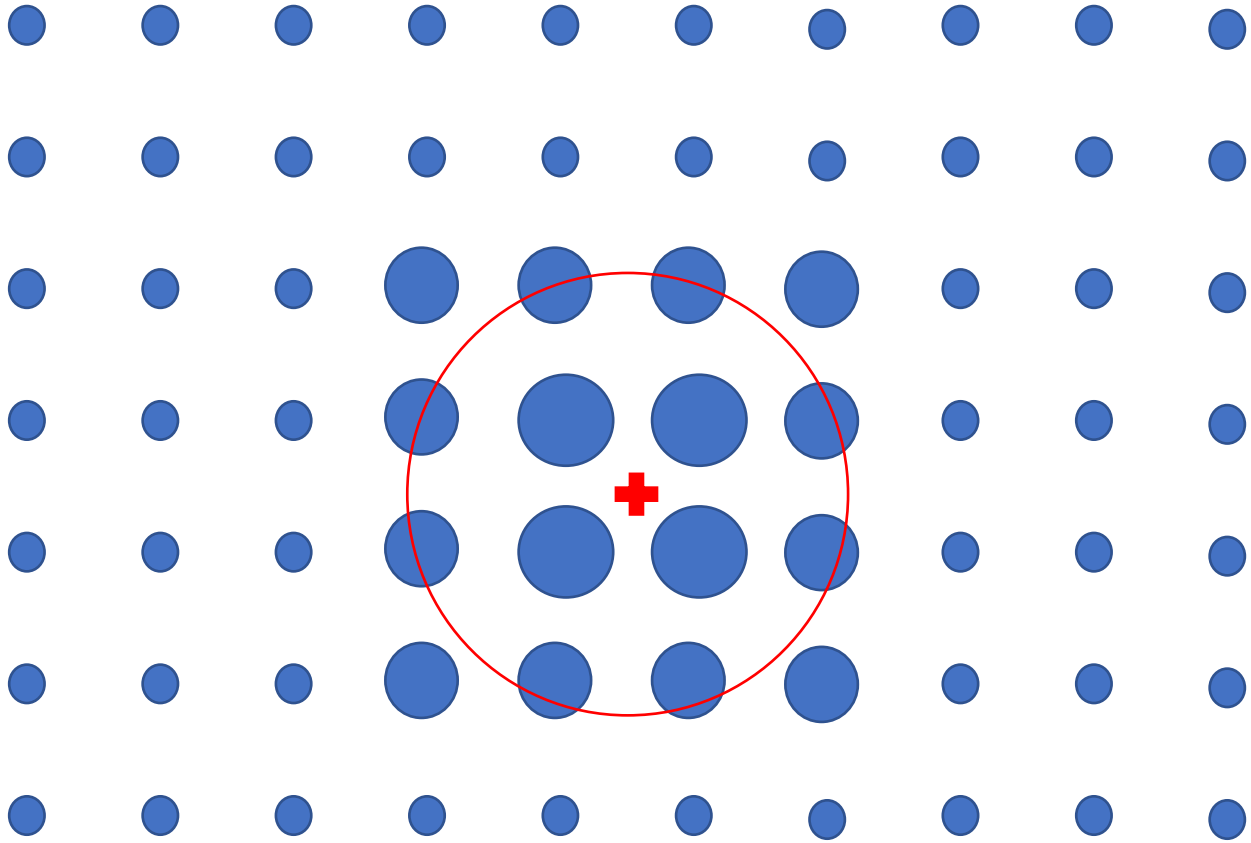
Mean Shift with Images

- Weight is assigned to each pixel points



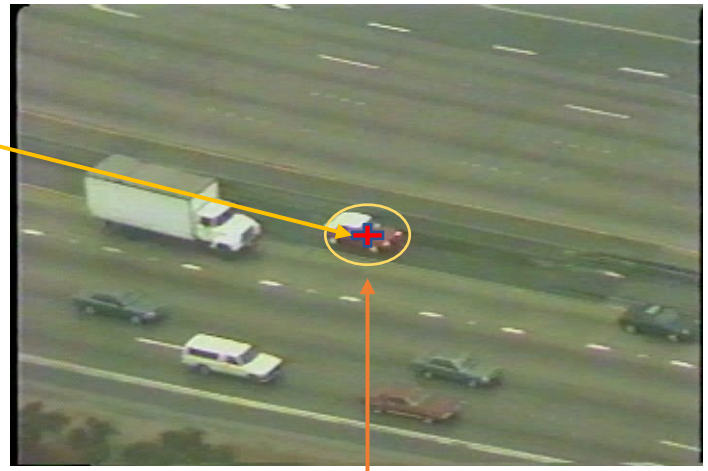
Mean Shift with Images

- Weight is assigned to each pixel points



Mean Shift with Video Tracking

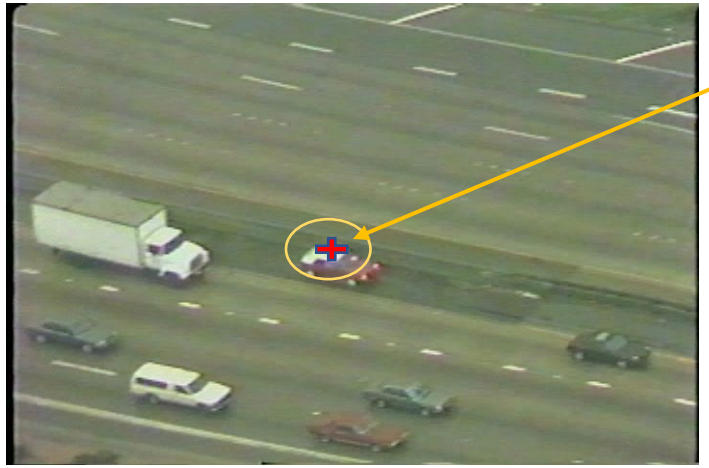
Frame 1



x centre of target

Target Selected in frame 1

Frame 2



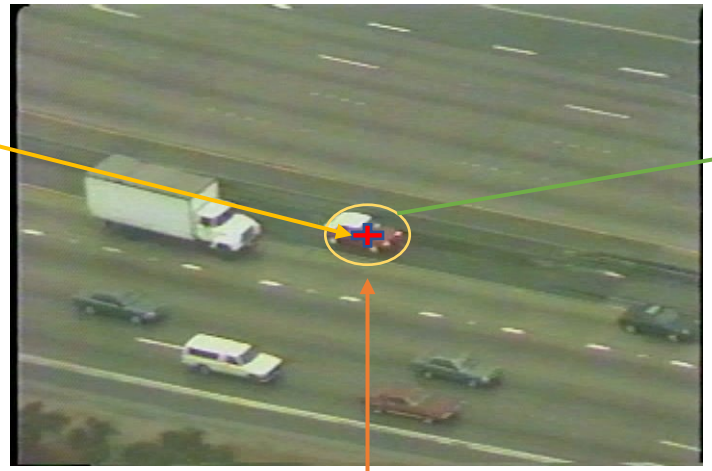
y centre of candidate

Goal: Use Mean-shift algorithm to find the best candidate location in frame 2

Step1: Target Modelling

Frame 1

x centre of target



Represented by m-dimensional vector

$$q = \{q_1, .., q_u\} \; where \; \sum_{u=1}^{m} q_u = 1$$

Target Selected in frame 1

Can be written as

Color ID

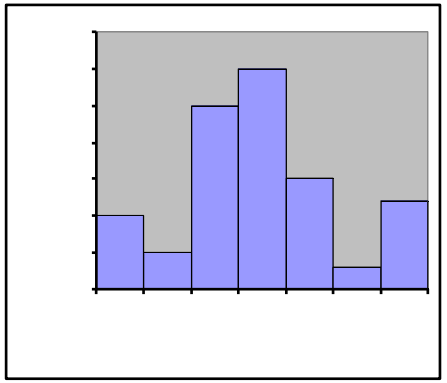
$$q_u = C \sum_{i=1}^{n} k(\|x_i\|^2)\delta[b(x_i) - u]$$

Normalization factor

Sum over all pixels

Weight function

Kronecker delta function



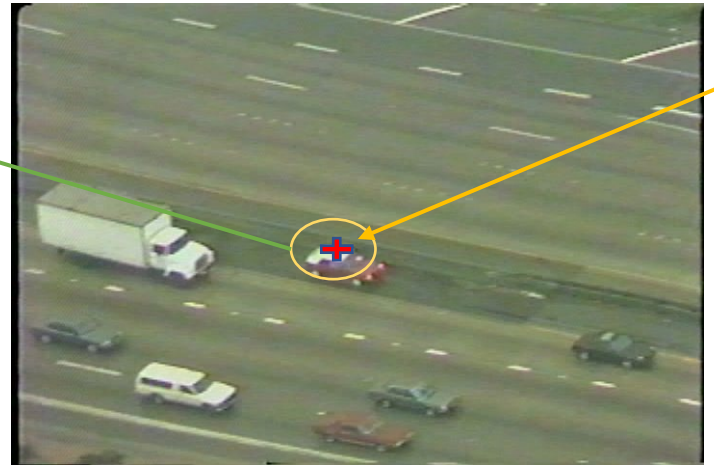
m-bin histogram

Step 2: Candidate Modelling

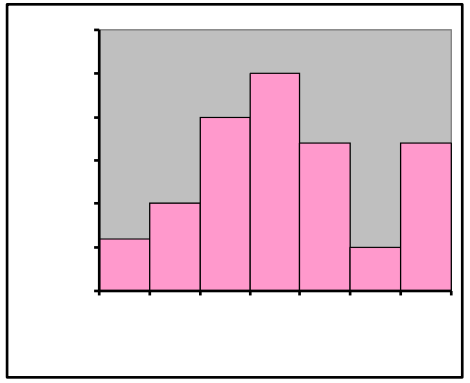
Represented by m-dimensional vector

$$p(y) = \{p(y), .., p_u(y)\}\ where \sum_{u=1}^{m} p_u(y) = 1$$

Frame 2



y centre of candidate



M bin histogram
(centred at y)

Using the same profile k and radius(bandwidth) h

$$p_u(\mathrm{y}) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{y-x_i}{h}\right\|^2\right) \delta[b(x_i) - u]$$

Normalization constant $C_h = \frac{1}{\sum_{i=1}^{n_h} k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)}$

Step 3: Measure Similarity Between target and candidate

- How to measure the similarity between two vectors?

$$Similarity\ (p(\boldsymbol{y}) = \{\boldsymbol{p}(y), .., p(y)_u\}, \boldsymbol{q} = \{q_1, .., q_u\}$$

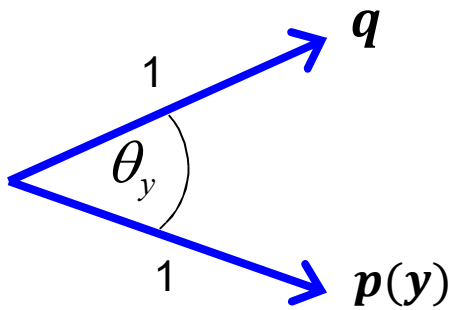
- Bhattacharyya Coefficient

$$\rho(\boldsymbol{y}) = \rho[\boldsymbol{p}(y), \boldsymbol{q}] = \sum_{u=1}^{m} \sqrt{p_u(y) q_u}$$

What is Bhattacharyya Coefficient?

- Just a cosine distance between two unit vectors

$$\rho(\boldsymbol{y}) = cos\theta_{\boldsymbol{y}} = \frac{\boldsymbol{p}(\boldsymbol{y})^T \boldsymbol{q}}{\|\boldsymbol{p}\|\|\boldsymbol{q}\|} = \sum_u \sqrt{p_u(\boldsymbol{y}) q_u}$$



- Distance function becomes

$$d(\boldsymbol{y}) = \sqrt{1 - \rho[\boldsymbol{p}(y), \boldsymbol{q}]}$$

Step 4: Objective function

- Objective: maximize the similarity function

$$\max_{\boldsymbol{y}} \rho[\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}]$$

$$p_u(\mathrm{y}) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{y-x_i}{h}\right\|^2\right) \delta[\mathrm{b}(x_i) - u]$$

By Taylor Expansion around $\boldsymbol{y}_o$

$$\rho[\boldsymbol{p}(y), \boldsymbol{q}] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{p_u(\boldsymbol{y_o})q_u} + \frac{1}{2}\sum_{u=1}^{m} p_u(\boldsymbol{y})\sqrt{\frac{q_u}{p_u(\boldsymbol{y_o})}}$$

$$w_i = \sum_{i=1}^{n_h} \delta[\mathrm{b}(x_i) - u]\sqrt{\frac{q_u}{p_u(\boldsymbol{y_o})}}$$

Then we have

$$\rho[\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{p_u(\boldsymbol{y_o})q_u} + \frac{C_h}{2}\sum_{i=1}^{n_h} w_i k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)$$

Independ of $\boldsymbol{y}$

Maximize this term

Objective Function

$$\rho[\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{p_u(\boldsymbol{y_0})q_u} + \frac{C_h}{2}\sum_{i=1}^{n_h} w_i k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)$$

Independend of $\boldsymbol{y}$

Maximize this term

$$\max_y \rho[\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}] = \max_y \frac{C_h}{2}\sum_{i=1}^{n_h} w_i k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)$$

Mean-Shift

Updated Candidate position:

$$\boldsymbol{y_1} = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\|\frac{y_0-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{y_0-x_i}{h}\right\|^2\right)}$$

Finally: Mean-Shift Object Tracking Algorithm

For each frame

1 Initialize the location $\boldsymbol{y_0}$
compute $\boldsymbol{q}$ and $\boldsymbol{p(y_0)}$

2 Evaluate $\rho[\boldsymbol{p(y_0)}, \boldsymbol{q}]$

3 Derive Weight $\{w_i\}_{i=1\ldots n_h}$

4 Compute and shift to $\boldsymbol{y_1}$

5 Update $\boldsymbol{p(y_1)}$ and $\rho[\boldsymbol{p(y_1)}, \boldsymbol{q}]$

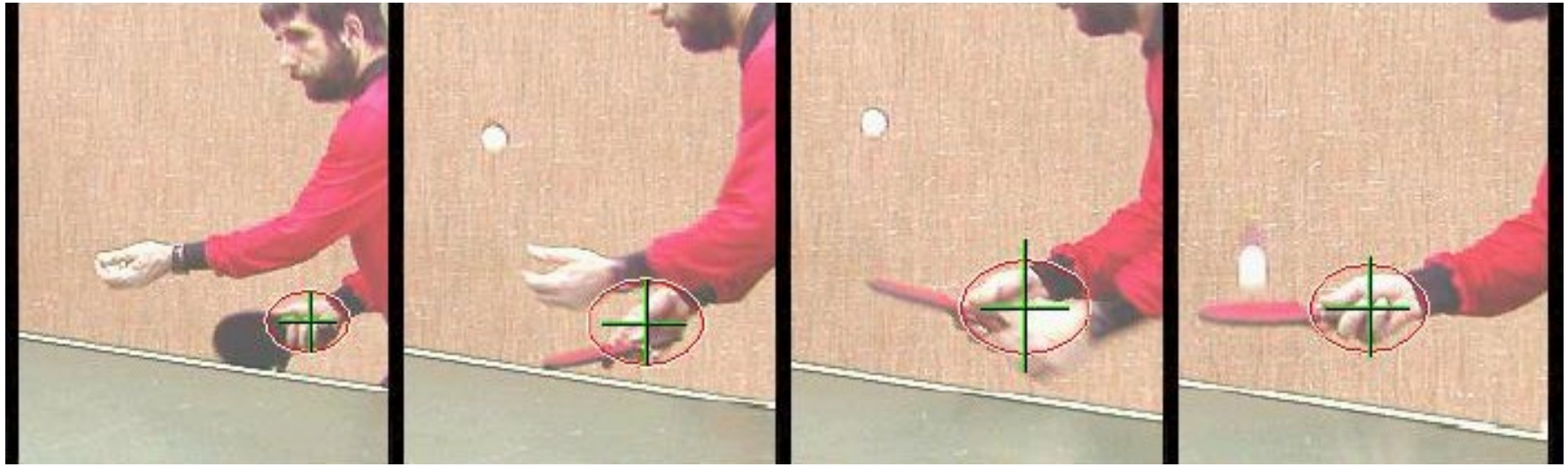
6 while $\rho[\boldsymbol{p(y_1)}, \boldsymbol{q}] < \rho[\boldsymbol{p(y_1)}, \boldsymbol{q}]$
$\boldsymbol{y_1} \leftarrow \frac{\mathbf{1}}{\mathbf{2}}(\boldsymbol{y_1} + \boldsymbol{y_0})$

7 if $\|\boldsymbol{y_1} - \boldsymbol{y_0}\| <\in \varepsilon$ stop
otherwise $\boldsymbol{y_0} \leftarrow \boldsymbol{y_1}$ goto 1

$$w_i = \sum_{i=1}^{n_h} \delta[\mathrm{b}(x_i) - u] \sqrt{\frac{q_u}{p_u(\boldsymbol{y_o})}}$$

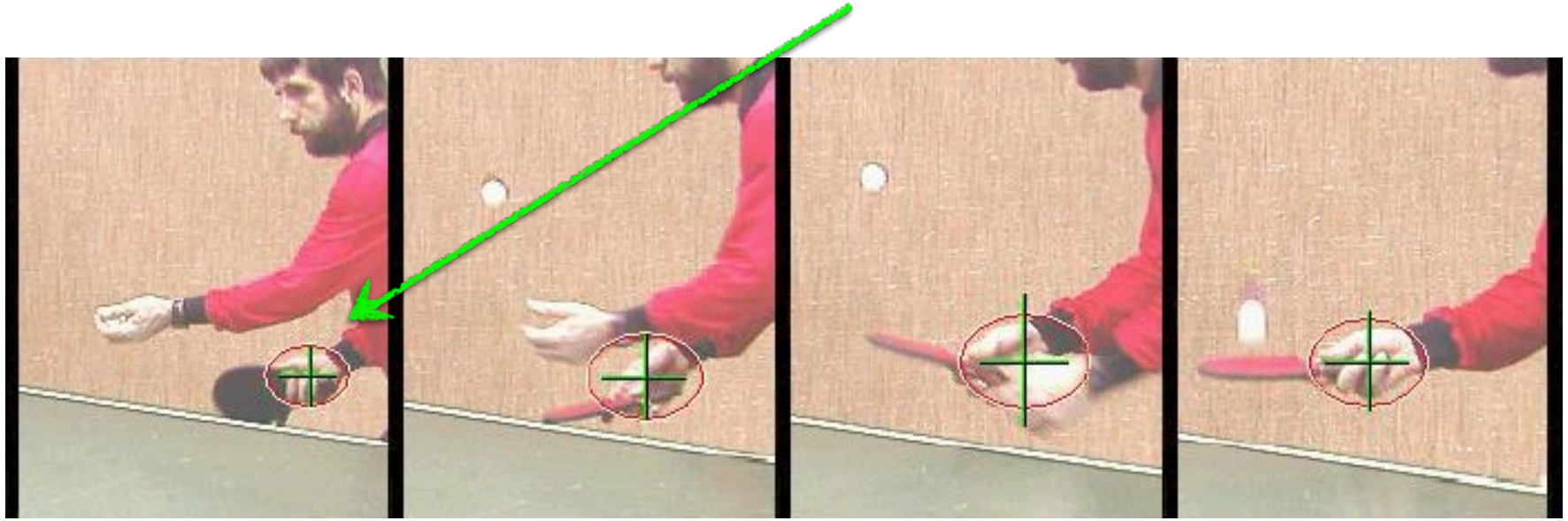
$$\boldsymbol{y_1} = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\|\frac{y_0 - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{y_0 - x_i}{h}\right\|^2\right)}$$

Non-rigid object tracking



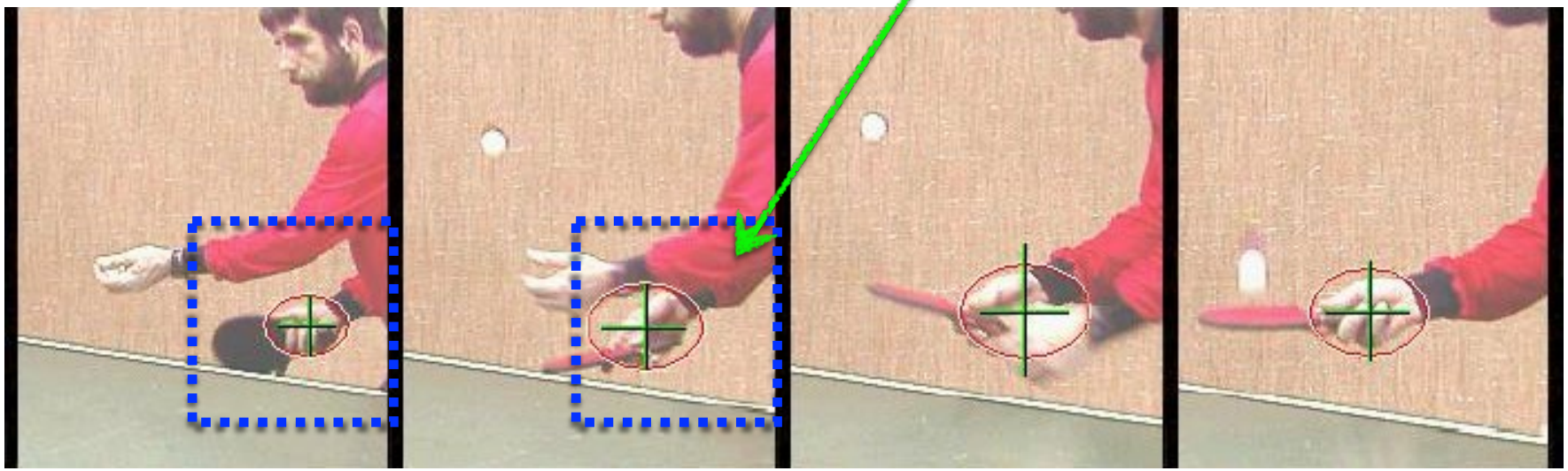
hand tracking

Compute a descriptor for the target



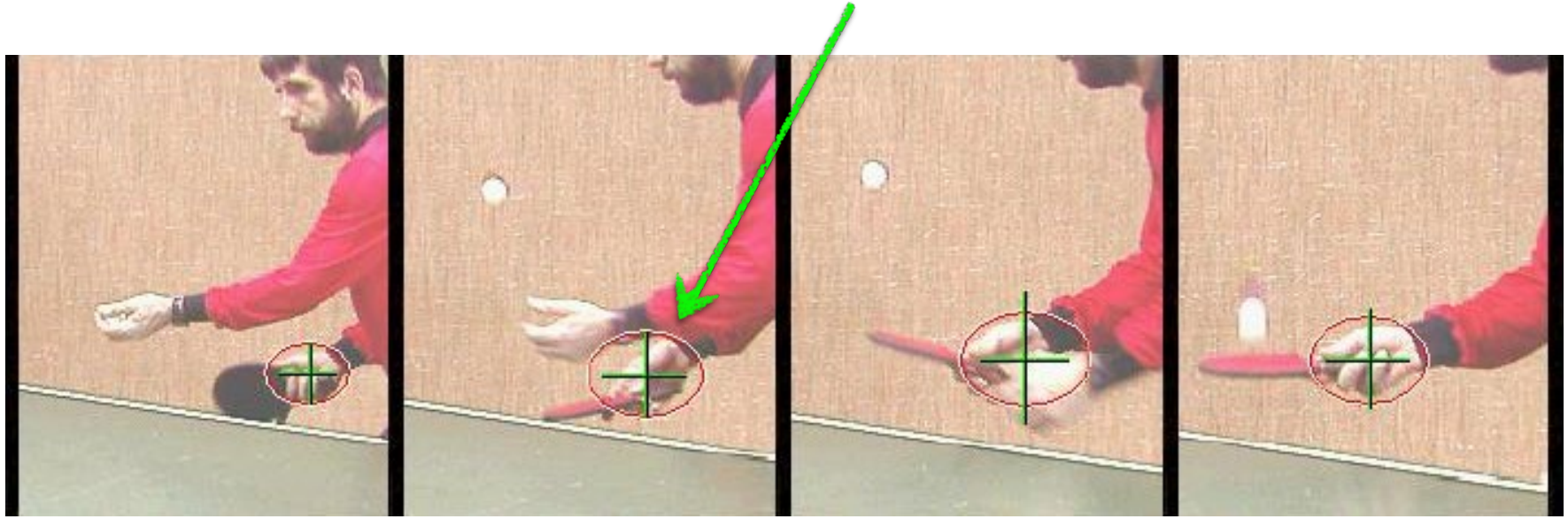
Target

Search for similar descriptor in neighborhood in next frame



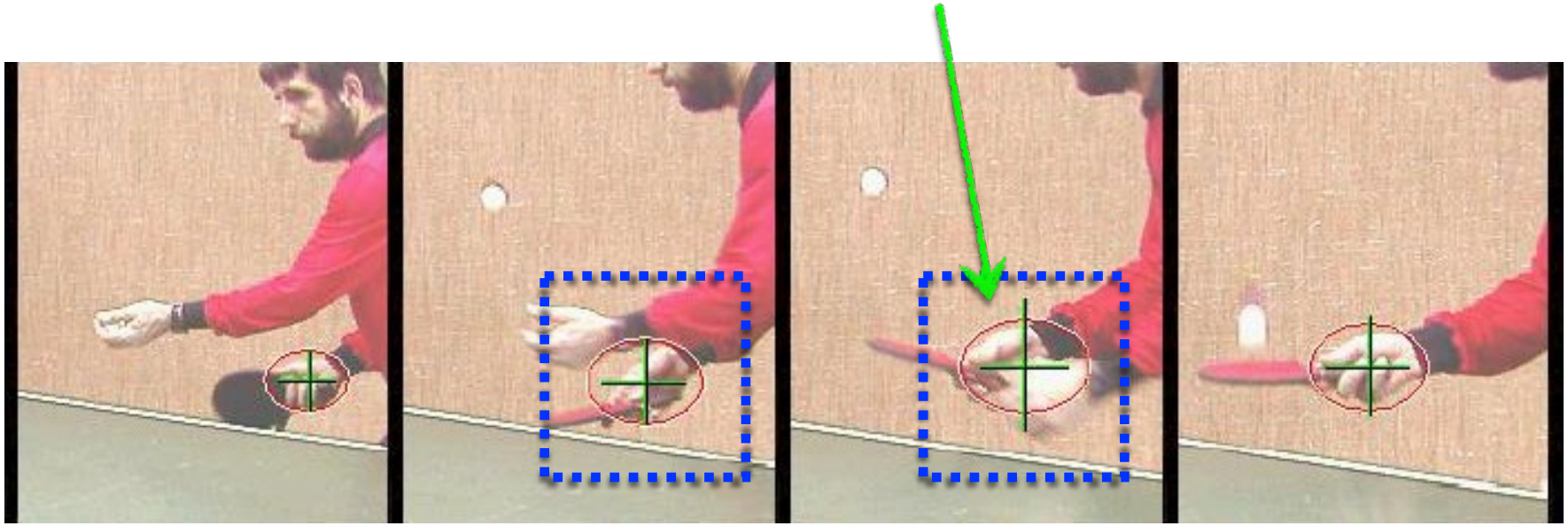
Target Candidate

Compute a descriptor for the new target

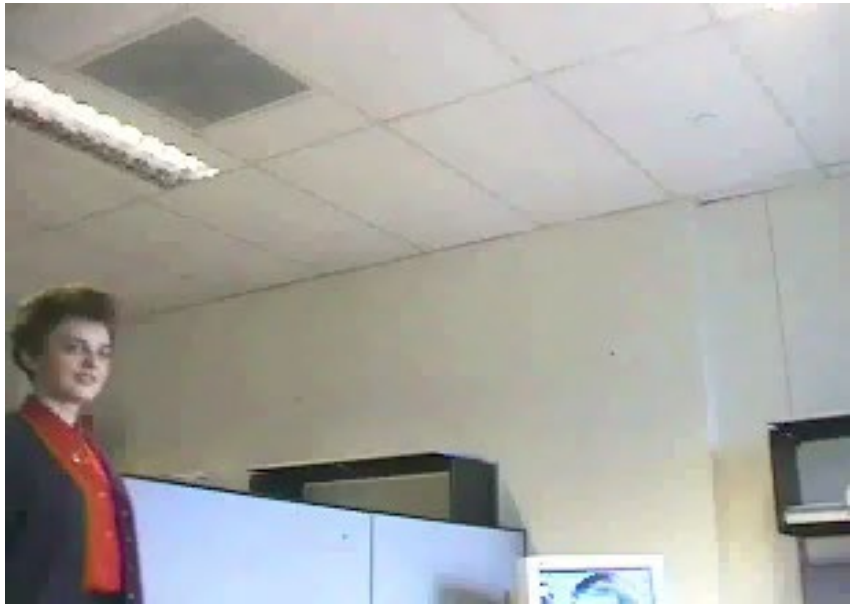


Target

Search for similar descriptor in neighborhood in next frame



Target Candidate



Slide Source: //www.cs.cmu.edu

More Readings

- SIMON BAKER AND IAIN MATTHEWS, “**Lucas-Kanade 20 Years**
- **On: A Unifying Framework**”, IJCV, 2004.
- Section 8.2, Richard Szeliski, "Computer Vision: Algorithms and Application".

Some Implementation Samples

- Some Matlab Implementation: Lucas Kanade with Pyramid http://www.mathworks.com/matlabcentral/fileexchange/30822

- Affine tracking: http://www.mathworks.com/matlabcentral/fileexchange/24677-lucas-kanade-affine-template-tracking