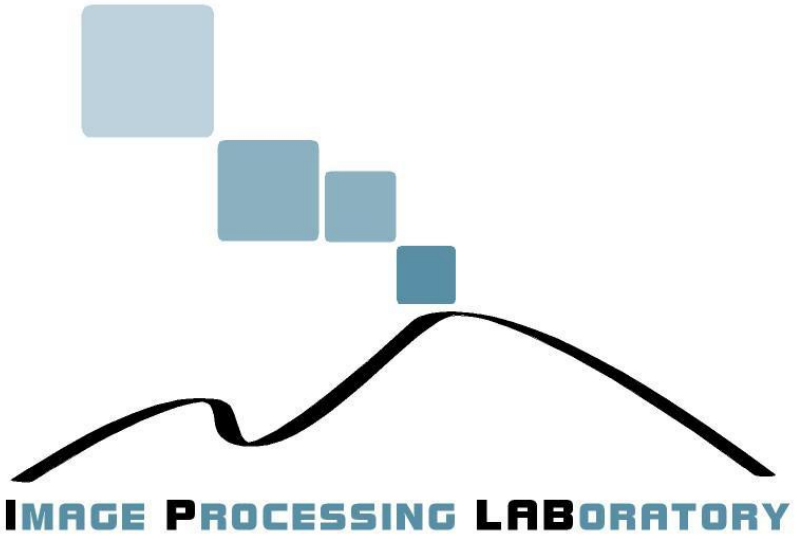
Visual Tracking

Image Processing Laboratory
Dipartimento di Matematica e Informatica
Università degli studi di Catania



What is visual tracking?

“estimation of the target location over time”

Visual Tracking applications

Six main areas:

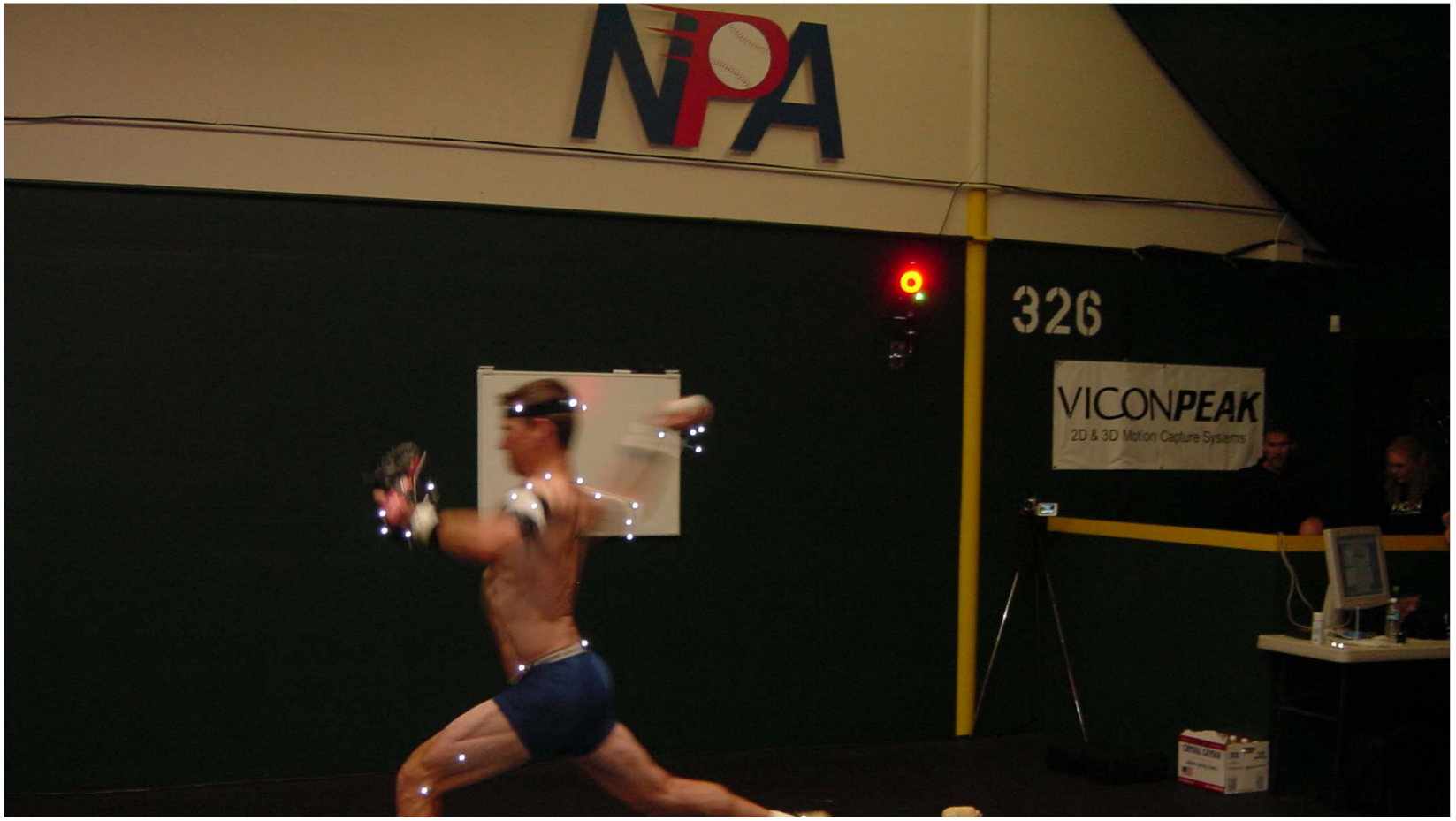
- Media production and augmented reality
- Medical applications and biological research
- Surveillance and business applications
- Robotics and unmanned vehicles
- Tele-collaboration and interactive gaming
- Art installations and performances

Visual Tracking Applications

motion capture: videogames, movies

Visual Tracking Applications

performance analysis: sport

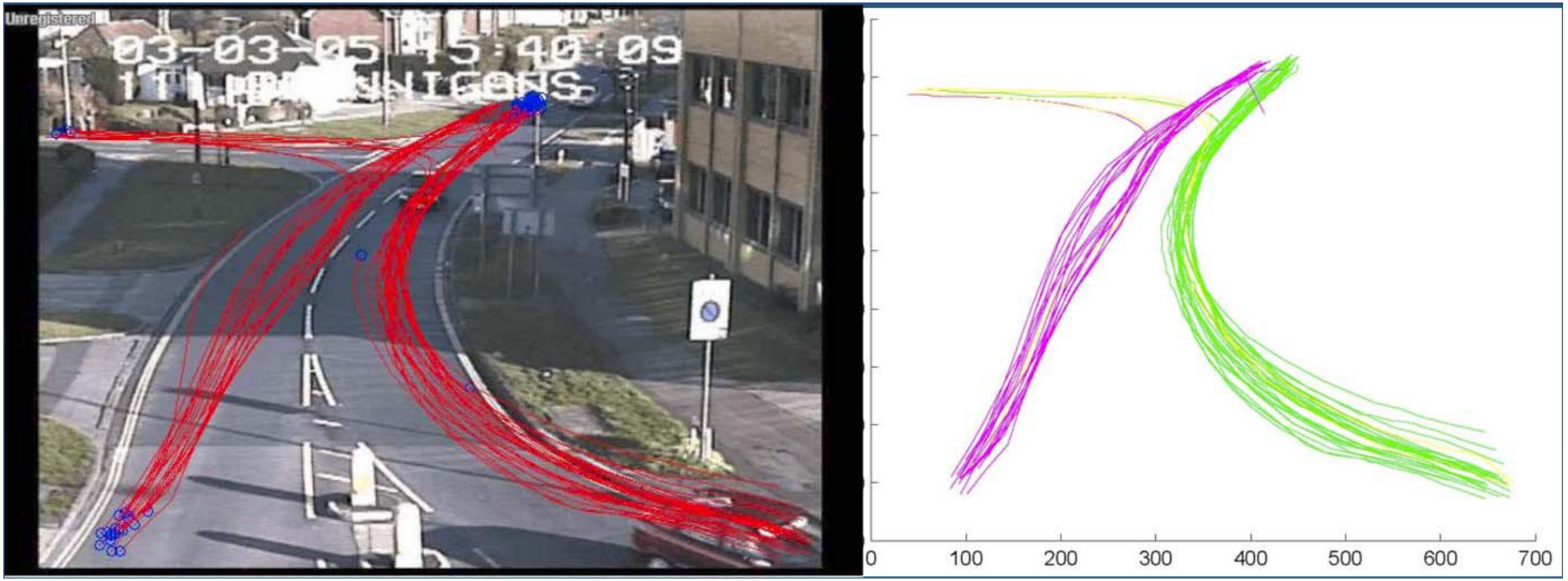


Visual Tracking Applications

video surveillance

Visual Tracking Applications

traffic monitoring



Problem Formulation

"estimation of the target state over time"

Let $\mathbf{I} = \{I_k : k \in \mathbb{N}\}$ be the frames of a video sequence, where $I_k \in E_I$ is the frame at time k and E_I is the space of all possible images.

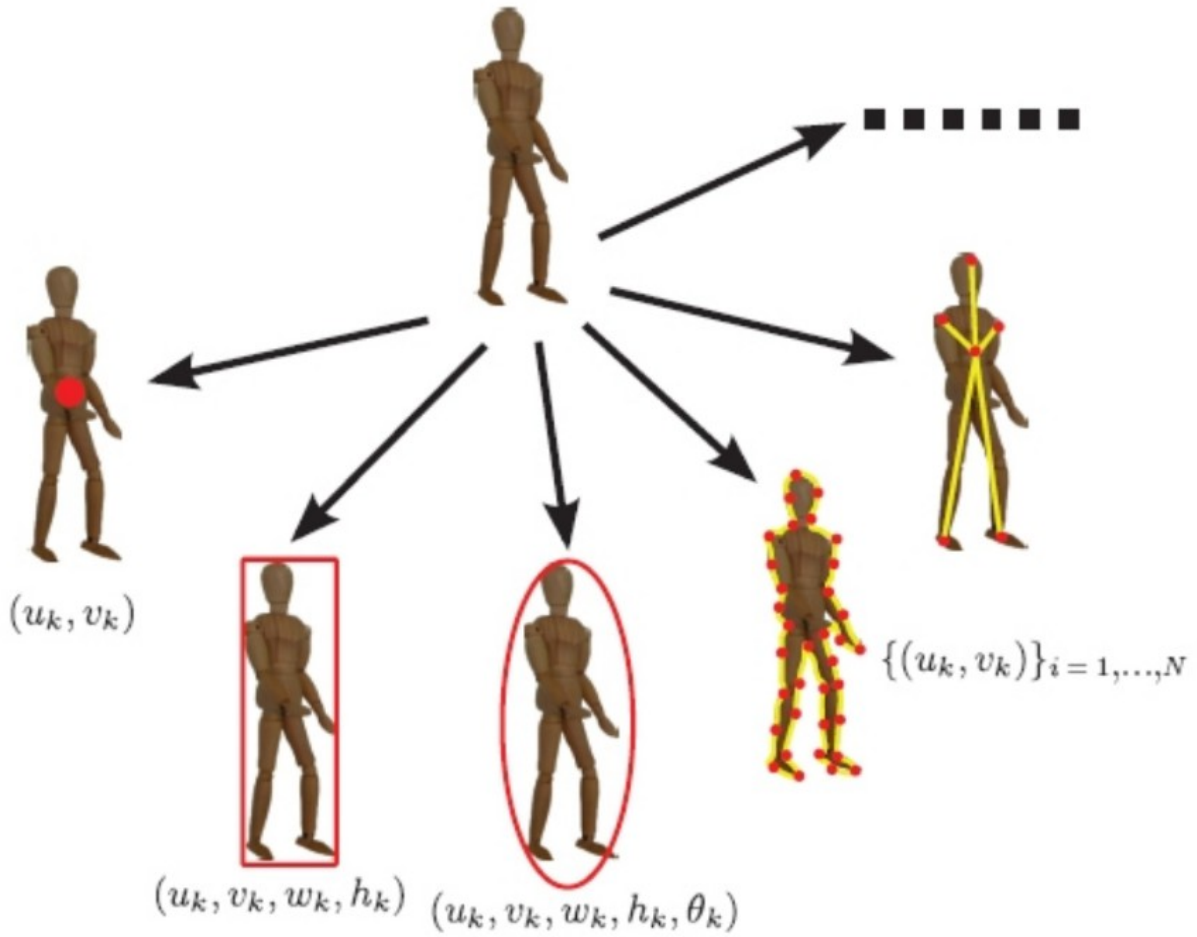
Tracking a single target object can be formulated as the estimation of the time series $\mathbf{x} = \{x_k : k \in \mathbb{N}\}$ over a set of discrete time instants indexed by k based on the information in $\mathbf{I}$.

The vectors $x_k \in E_s$ are the **states** of the target and E_s is the state space. The time series $\mathbf{x}$ is also known as the trajectory of the target in E_s .

The information encoded in the state x_k depend on the application.

Problem Formulation

object state



Problem Formulation

feature extraction

"estimation of the target state over time"

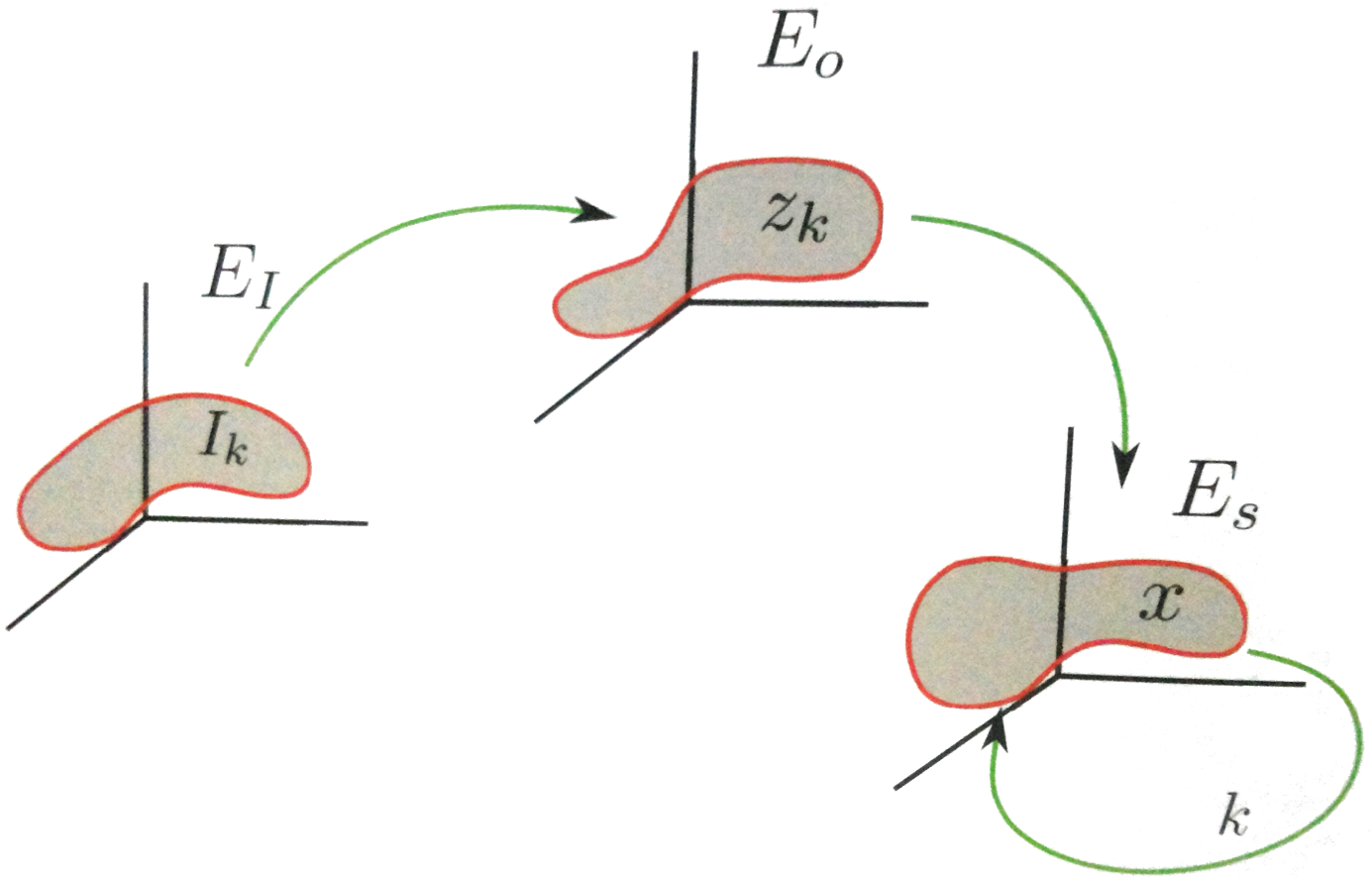
I_k may be mapped onto a feature (or observation) space E_o that highlights information relevant to the tracking problem. The observation generated by a target is encoded in $z_k \in E_o$.

The operations that are necessary to transform the image space E_I to the observation space E_o are referred to as **feature extraction**.

Visual trackers propagate the information in the state x_k over time using the extracted features. A localization strategy defines how to use the image features to produce an estimate of the target state x_k .

Problem Formulation

feature extraction



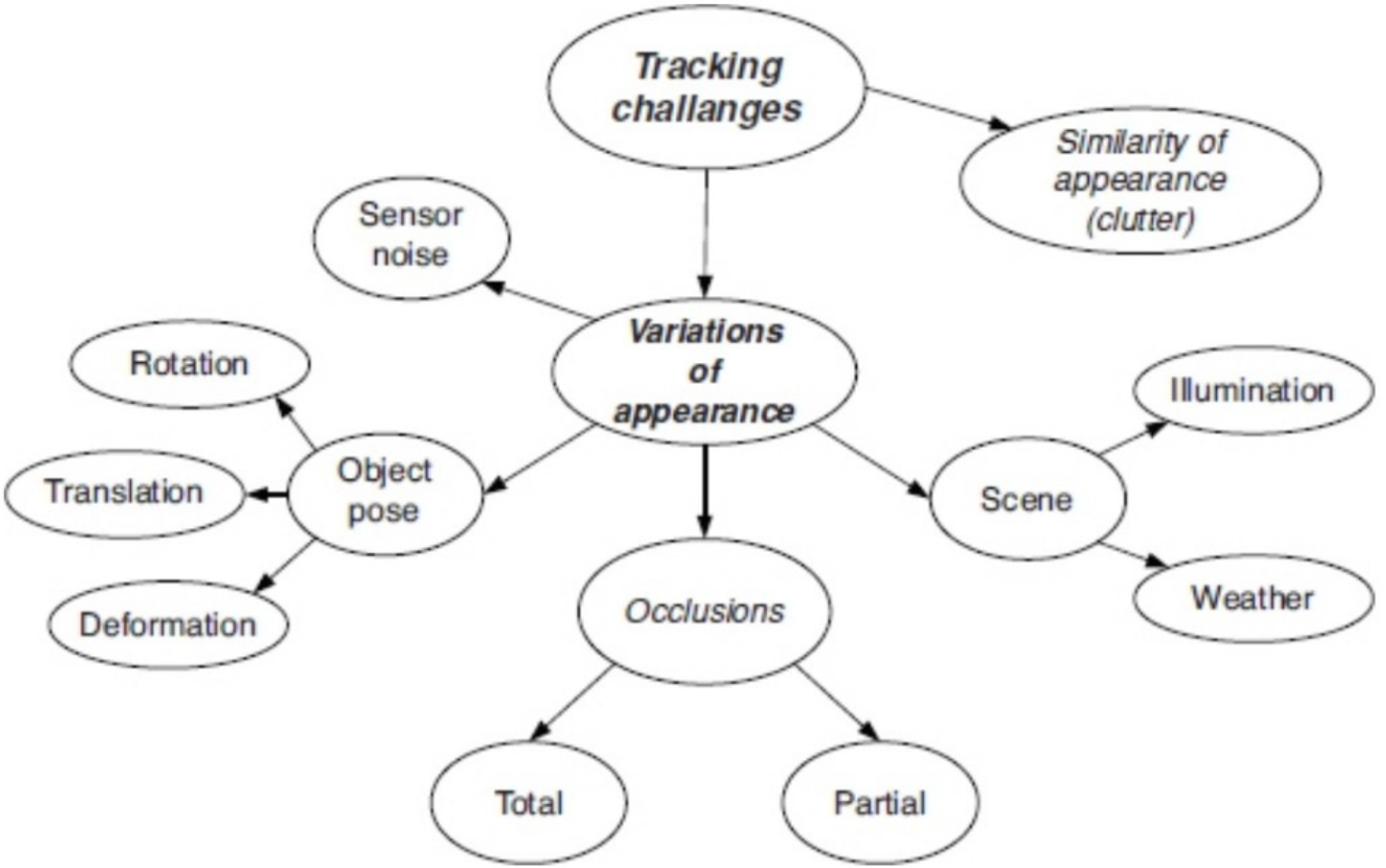
E_I : space of all possible images

E_o : feature or observation space

E_s : state space

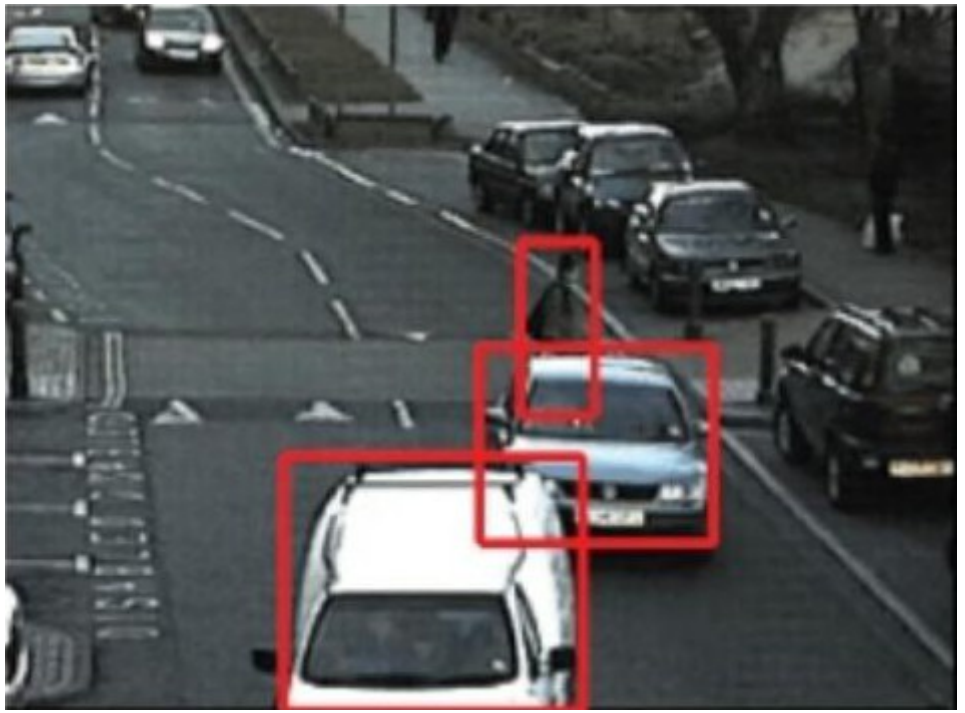
k : time index

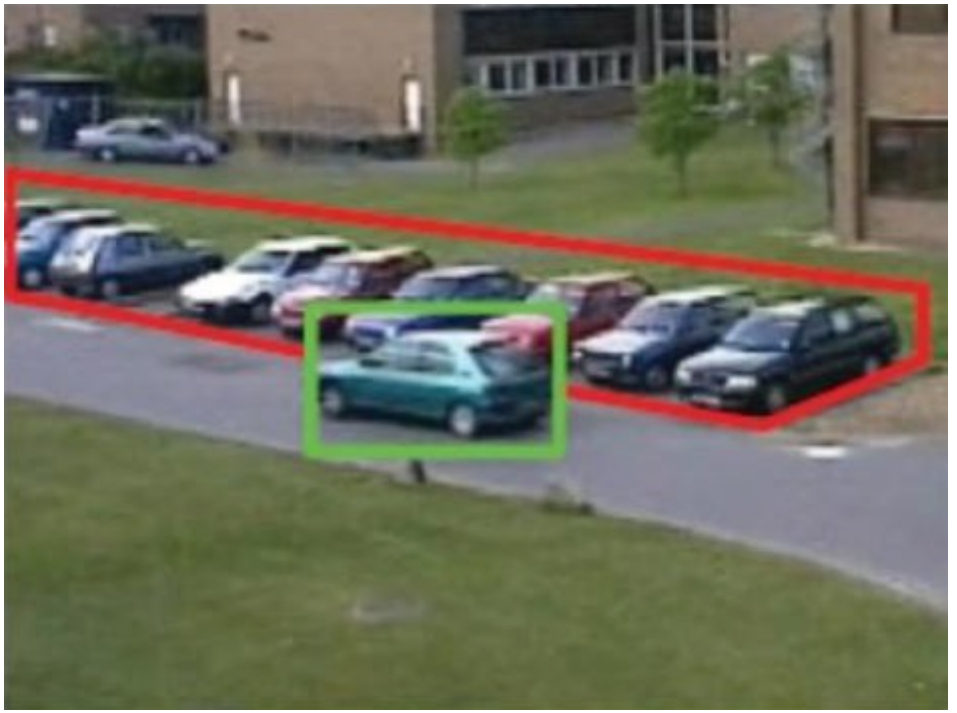
Visual Tracking Challenges



Visual Tracking Challenges

occlusion - clutter

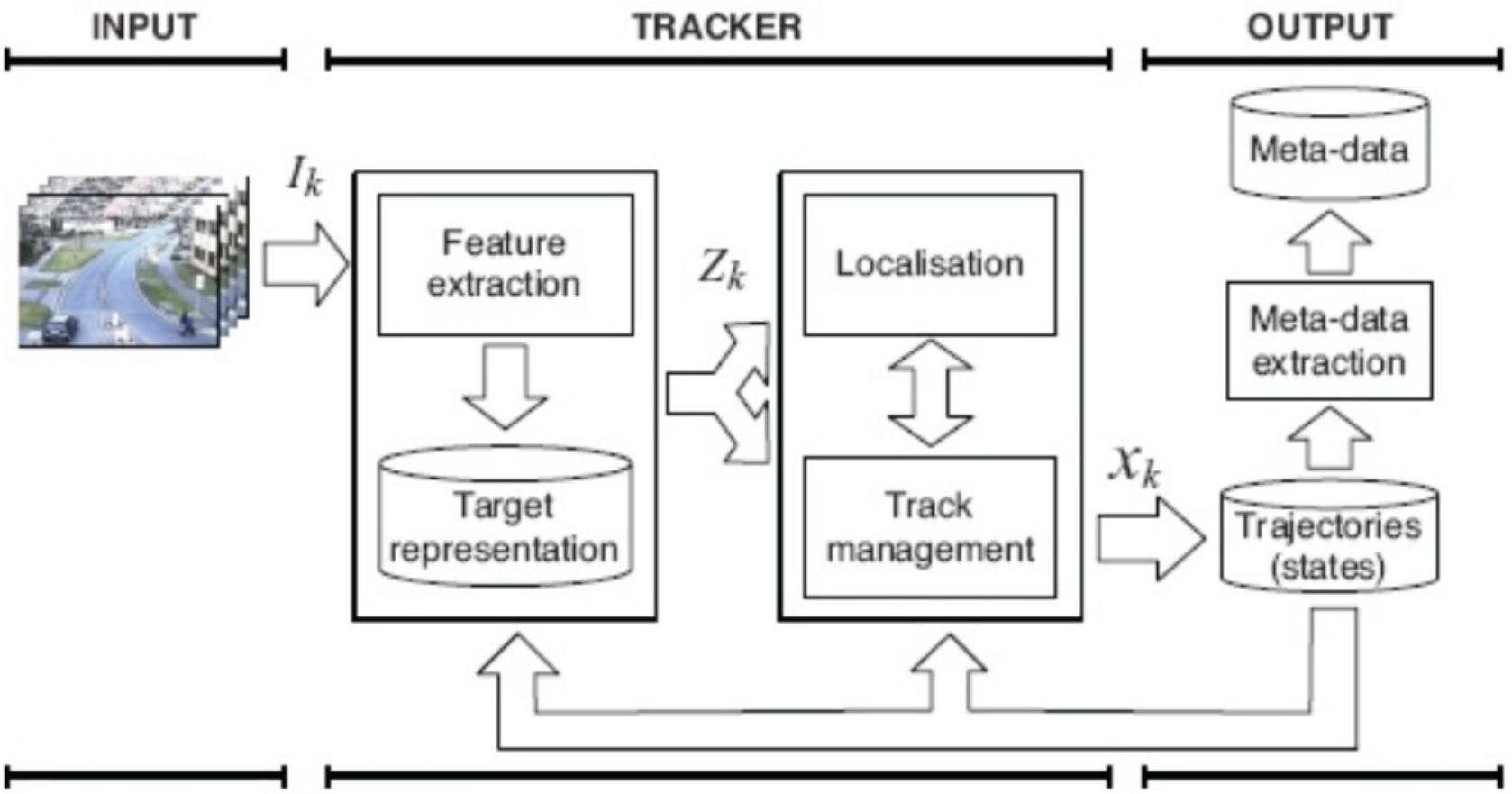




Visual Tracking Challenges

change of appearance

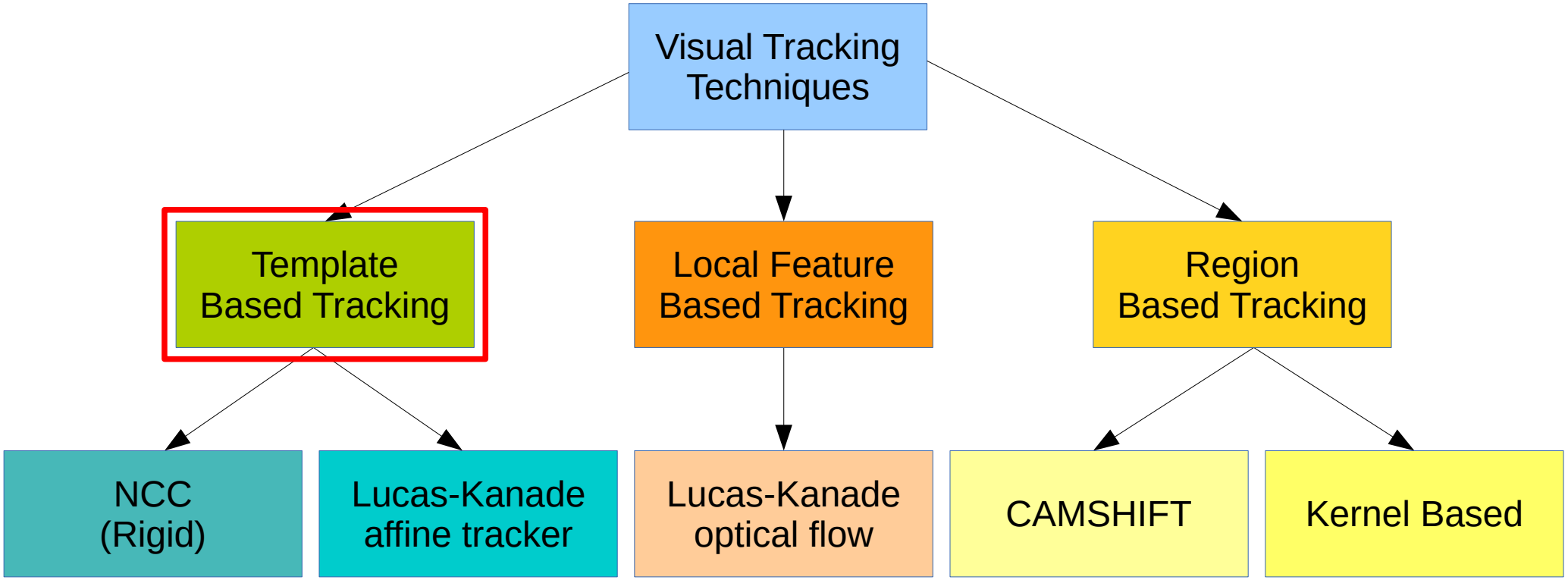
Visual Tracking Pipeline



I_k : video frames Z_k : observations x_k : object states

Visual Tracking Techniques

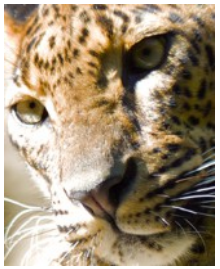
Visual tracking techniques can be characterized by the way the object is **represented** and the way the object is **located**

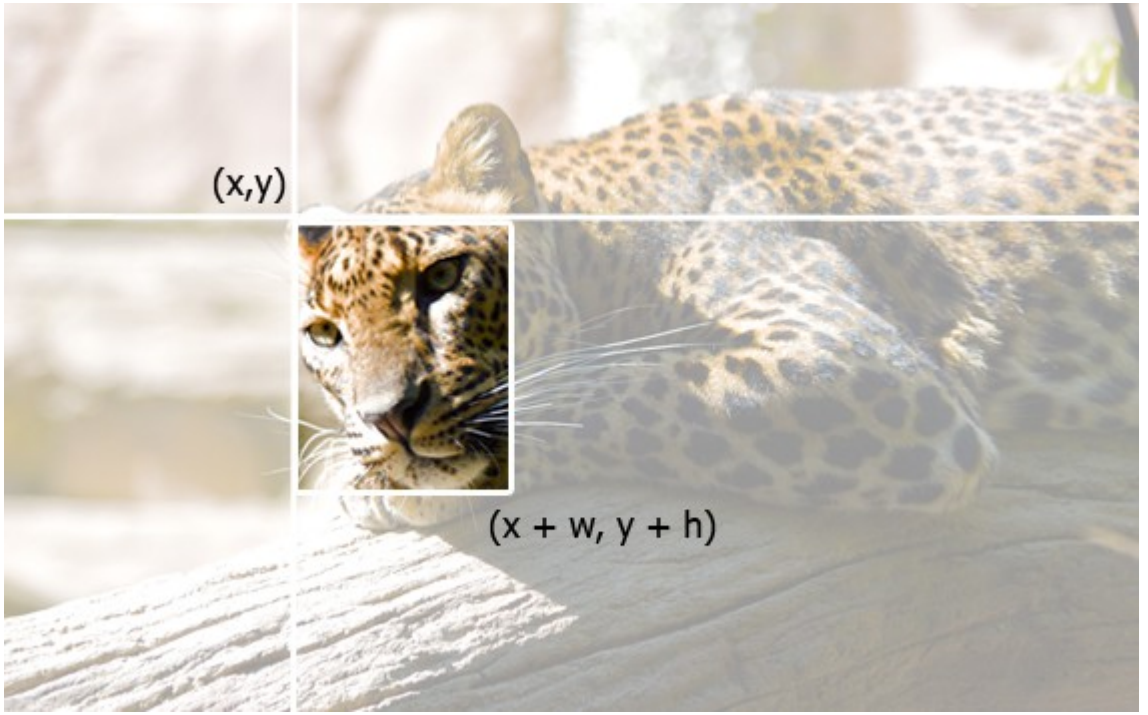


Template Based Tracking

Objects are **represented** as image patches and **located** by maximizing some *similarity measure* between the *template object* and a series of *candidates* extracted from the current frame (template matching).

target object

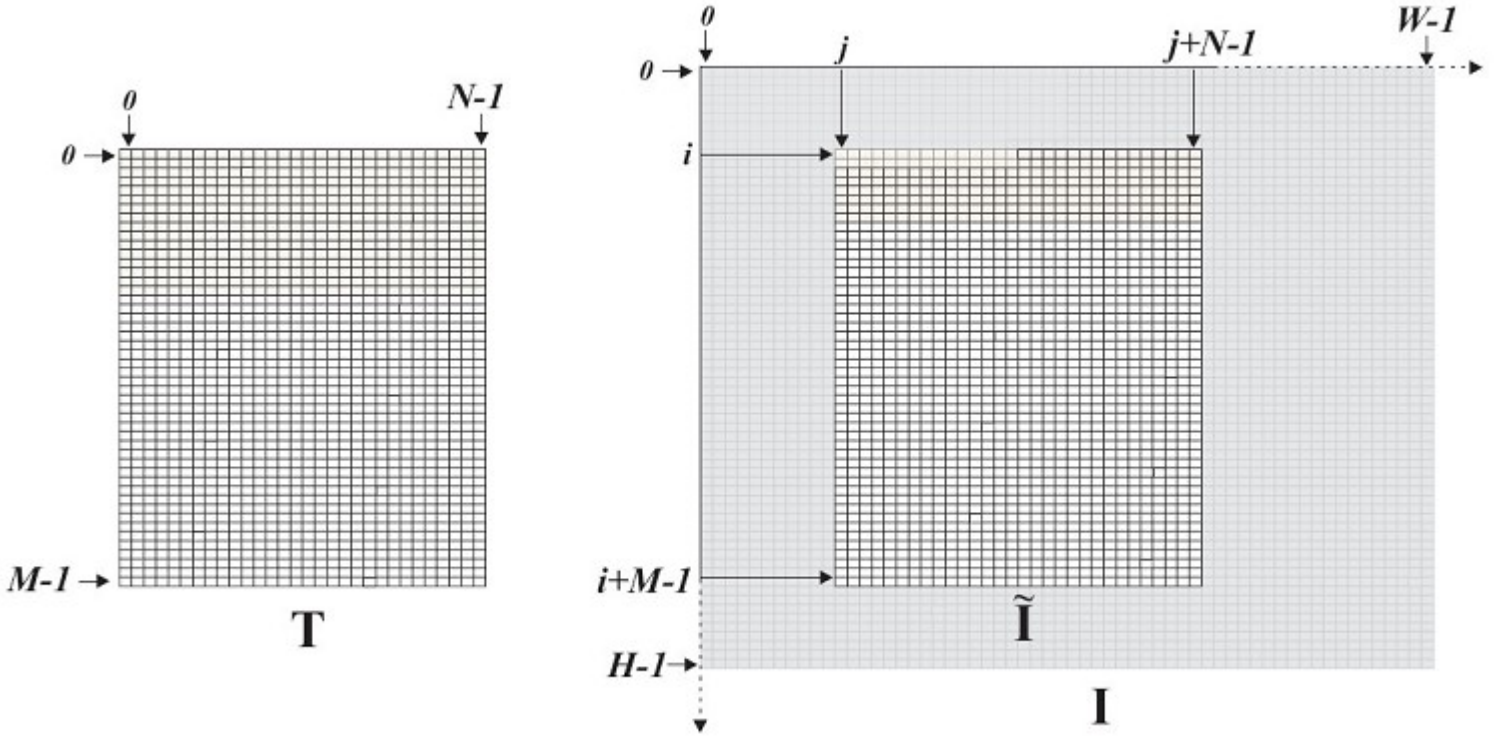




where «w,h» are the template dimensions and «x,y» is the considered sub-image position within the frame.

Similarity Measures

In order to measure the similarity degree between the target object and the candidates we need a **similarity measure**.



Where:

- T is the template (*target object*)
- $M \times N$ are the template dimensions
- $\tilde{I}$ is the sub-image (*candidate*)
- (i, j) is the location of the sub-image

Similarity measures:

Sum of Absolute Differences (**SAD**)
Sum of Squared Differences (**SSD**)
Normalized Corss Correlation (**NCC**)

Similarity Measures

sum of absolute differences (SAD)

$$SAD(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |I(i+m, j+n) - T(m,n)|$$

- A perfect match corresponds to 0
- SAD represents the L_1 norm between $I(i+m, j+n)$ and $T(m,n)$.

Similarity Measures

sum of squared differences (SSD)

$$SSD(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (I(i+m, j+n) - T(m,n))^2$$

- A perfect match corresponds to 0
- SSD is the euclidean norm (L_2) between $I(i+m, j+n)$ and $T(m,n)$.

Similarity Measures

normalized cross correlation (NCC)

$$NCC(i,j) = \frac{1}{N \cdot M} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \frac{(I(i+m, j+n) - \overline{I}_{i,j}) \cdot (T(m,n) - \overline{T})}{\sigma_{I_{i,j}} \cdot \sigma_T}$$

where $\overline{I}_{i,j}$ and $\overline{T}$ are the subimage and template image mean values respectively

- A perfect match corresponds to 1
- The worst match corresponds to -1 (max decorrelation)
- NCC corresponds to the cosine angle between $T(m,n)$ and $I(i+m, j+n)$

Similarity Measures

NCC explained

$$NCC(i,j) = \frac{1}{N \cdot M} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \frac{(I(i+m, j+n) - \overline{I}_{i,j}) \cdot (T(m,n) - \overline{T})}{\sigma_{I_{i,j}} \cdot \sigma_T}$$

mean mean variance variance

Fixed the (m,n) point, let $v_1 = I(i+m, j+n) - \overline{I}_{i,j}$ and $v_2 = T(m,n) - \overline{T}$ be two 3d vectors. The above product is: $\left\langle \frac{v_1}{||v_1||}, \frac{v_2}{||v_2||} \right\rangle$ where $||\cdot||$ is the L_2 norm and $\langle \cdot, \cdot \rangle$ is the inner product, a generalization of the dot product.

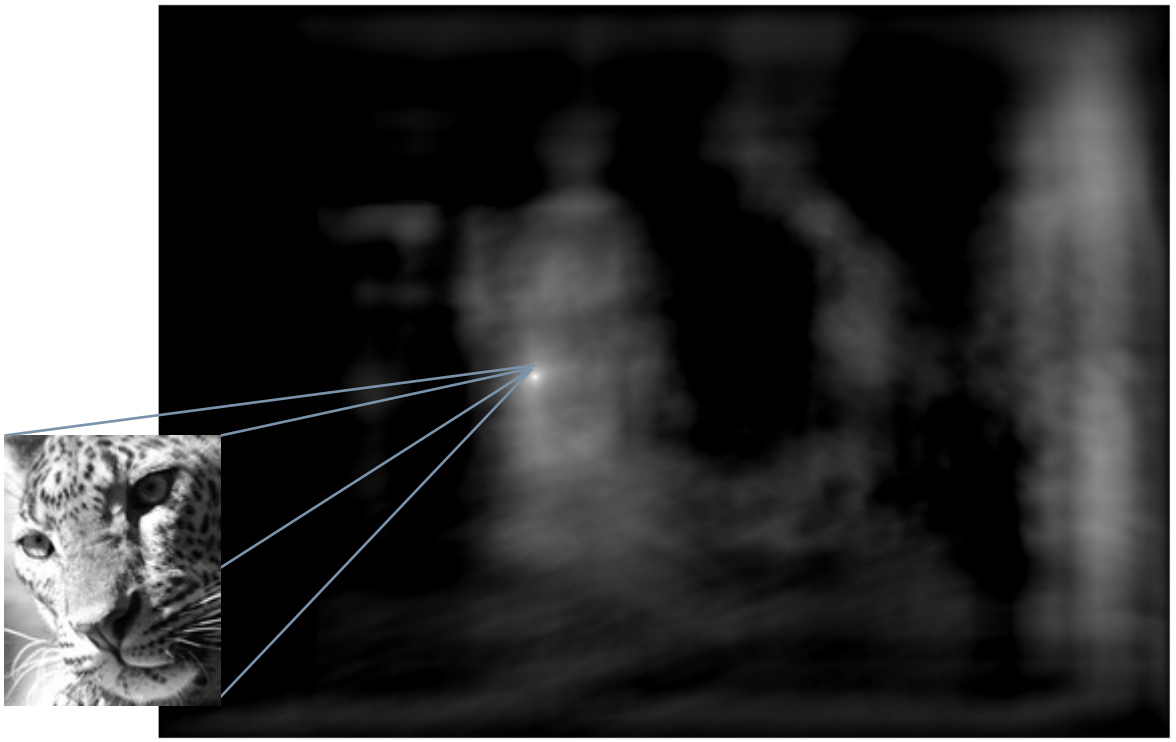
For a property of the inner product, if v_1 and v_2 are real vectors: $\cos\theta = \left\langle \frac{v_1}{||v_1||}, \frac{v_2}{||v_2||} \right\rangle$ being θ the angle between the two vectors.

$\cos\theta = 1$ only if v_1 equals v_2 multiplied by a positive scalar.

i.e., invariance to linear lighting changes

Template Matching

similarity measure example



Example of template matching using NCC as similarity measure

The similarity function maximum position is the object location

Template Matching

computational cost

Number of operations = $O((H \cdot W) \cdot (M \cdot N))$

where:

- H and W are the dimensions of the frame.
- M and N are the dimensions of the template.

The computational cost is proportional to the template and the frame dimensions.

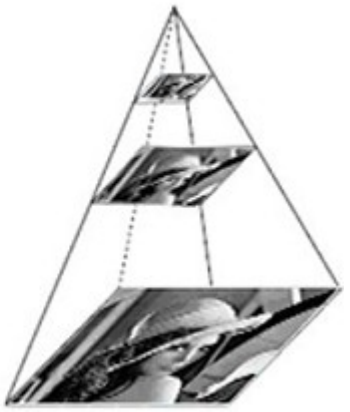
Possible Solutions:

- *Multi-Resolution*
- *SubTemplate*

Template Matching

computational cost - improvements

1. *Multi-Resolution*:



A low-resolution version of the template is first located in the lowest resolution image of the pyramids.

This defines a search area in the next level where to search for a higher resolution template.

At the last iteration the object is found in the original frame.

2. *Sub-Template*:

First a **sub-template** search is performed on the entire image. Next, only on those areas with a high similarity score, a regular search is performed.

Template Matching

limits

Limits:

- The template representation is not invariant to rotation, scale and other transformations.

So:

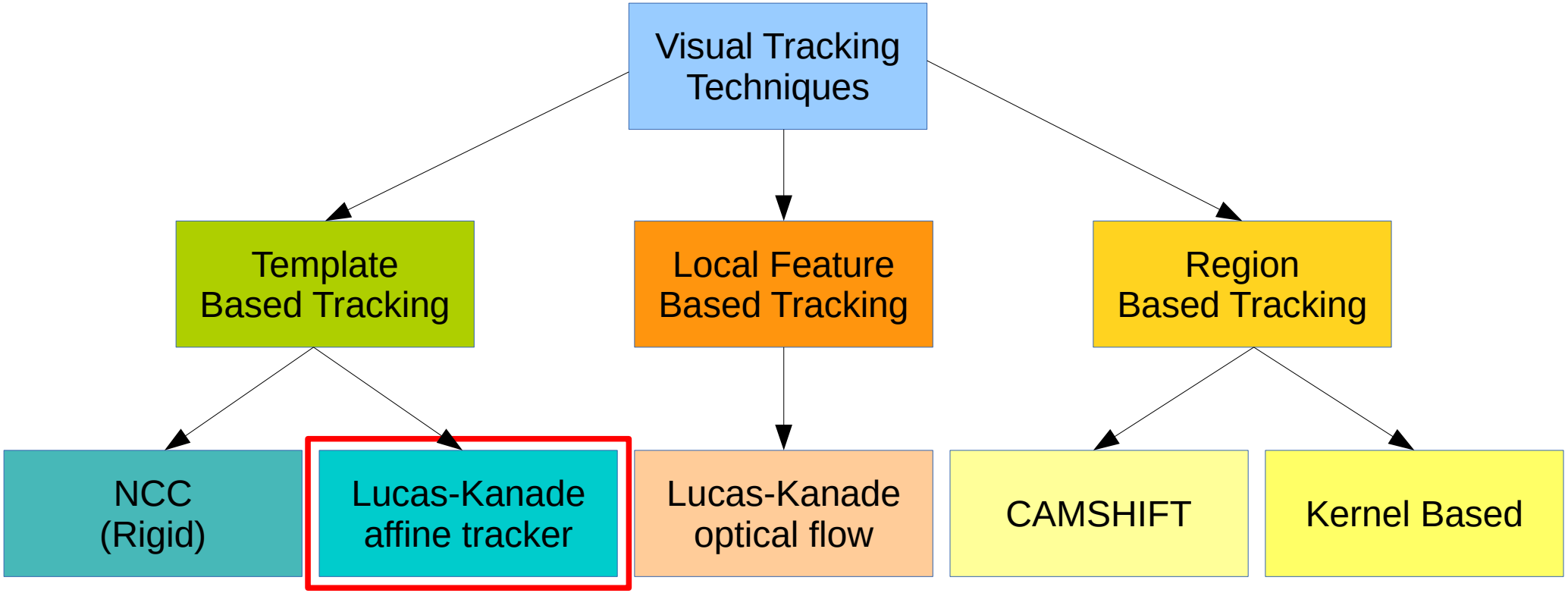
- The transformations have to be handled explicitly.

Possible solution:

- Estimating the transformation parameters during the search (i.e., search for warped versions of the template).

Lucas-Kanade registration algorithm

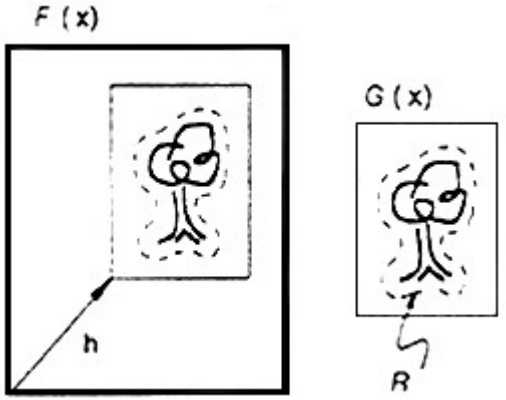
Visual Tracking Techniques



Lucas-Kanade Registration Algorithm

translational problem

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an input image.



The translational image registration problem can be characterized as follows: we are given functions $F(x)$ and $G(x)$ which give the respective pixel values at each location x in two images, where x is a vector. We wish to find the disparity vector h which minimizes some measure of the difference between $F(x+h)$ and $G(x)$ for x in some region of interest R.

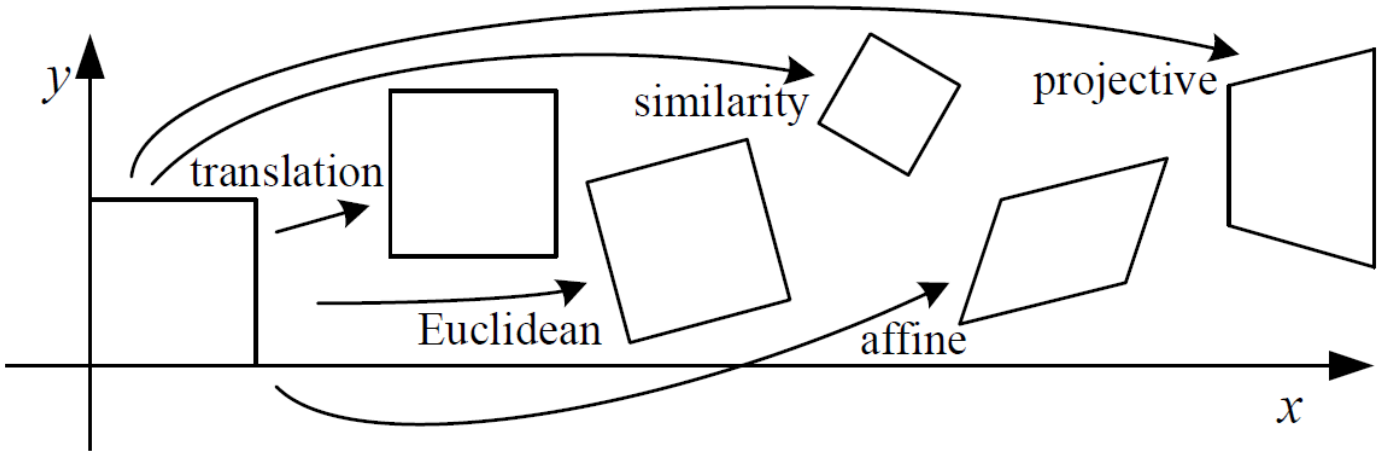
estimate the 2 translation parameters

Lucas-Kanade Registration Algorithm

general problem

We want to search for an affine-transformed version of the template estimating the 6 parameters of the transformation matrix:

$$M = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix}$$



estimate the 6 affine transformation parameters

Lucas-Kanade Registration Algorithm

the warp function

Let $W(x;p)$ be the deformation function of the x pixel using the vector parameters $p = (p_1, p_2, \ldots, p_n)^T$ from the template coordinate system to the image one.

The warp $W(x;p)$ takes the pixel x in the coordinate system of the template T and maps it to the sub-pixel location $W(x;p)$ in the coordinate system of the image I.

The W function can be a simple translation ($p = (p_1, p_2)$) or a 3D affine warp function ($p = (p_1, p_2, p_3, p_4, p_5, p_6)$). In the latter case $W(x;p)$ is defined as:

$$W(x;p) = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

In general the number of parameters n may be arbitrarily large and $W(x;p)$ can be arbitrarily complex.

Lucas-Kanade Registration Algorithm

goal

The aim of the algorithm is to minimize the *sum of squared errors* (**SSD**) between the template T and the image I warped back onto the coordinate system of the template:

$$\sum_x [I(W(x;p)) - T(x)]^2 \quad (1)$$

The minimization in (1) is performed with respect to the parameters vector p and the sum is performed over all of the pixels of the template T.

To optimize the expression in equation (1), the Lucas-Kanade algorithm assumes that a current estimate of the vector p is known and then iteratively solves for increments to the parameters Δp; i.e., the following expression is approximately minimized:

$$\sum_x [I(W(x;p+\Delta p)) - T(x)]^2 \quad (2)$$

with respect to Δp and then the parameters are updated:

$$p = p + \Delta p$$

These two steps are iterated until the estimates of the parameters p converge (Δp is under a fixed threshold). The initial parameters estimate: $p_0 = \vec{0}$ is often used.

the iterative minimization process is based on the gradient descent

Lucas-Kanade Registration Algorithm

derivation

The equation (1) can be expressed using the Taylor series expansion stopping at the first degree:

$$\sum_x \left[I(W(x;p)) + \bigtriangledown I \frac{\partial W}{\partial p} \Delta p - T(x) \right] \tag{3}$$

where $\bigtriangledown I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ is the gradient of the image I evaluated at $W(x;p)$ and $\frac{\partial W}{\partial p}$ is the warp Jacobian:

$$\frac{\partial W}{\partial p} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}$$

Calculating the derivative of the Taylor series expansion with respect to the parameter vector increment Δp and equating to zero, we obtain:

$$\Delta p = H^{-1} \sum_x \left[\bigtriangledown I \frac{\partial W}{\partial p} \right]^T [T(x) - I(W(x;p))] \tag{4}$$

where H is the hessian defined as:

$$H = \sum_x \left[\bigtriangledown I \frac{\partial W}{\partial p} \right]^T \left[\bigtriangledown I \frac{\partial W}{\partial p} \right] \tag{5}$$

Lucas-Kanade Registration Algorithm

algorithm

Iterate:

1. Warp I with $W(x;p)$ to compute $I(W(x;p))$
2. Compute the error image $T(x) - I(W(x;p))$
3. Warp the gradient ∇I with $W(x;p)$
4. Evaluate the Jacobian $\frac{\partial W}{\partial p}$ at $(x;p)$
5. Compute the steepest descent images $\nabla I \frac{\partial W}{\partial p}$
6. Compute the Hessian matrix using Equation (5)
7. Compute $\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(x;p))]$
8. Compute Δp using Equation (4)
9. Update the parameters $p = p + \Delta p$

until $||\Delta p|| < \varepsilon$

Computational cost of each iteration: $O(n^2 N + n^3)$

number of parameters

number of pixels in T

Lucas-Kanade Registration Algorithm

algorithm

Input:

Template T

Image I

Step 1 (image warp):

warp using the current estimate of p

Target candidate
$I(W(x;p))$

Step 2 (error image):

$T(x)$

$-$

$I(W(x;p))$

$=$

$T(x) - I(W(x;p))$

Lucas-Kanade Registration Algorithm

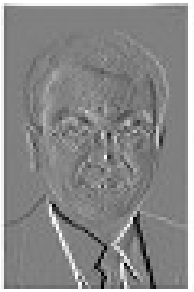
algorithm

Step 3 (gradient warp):

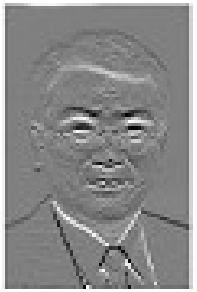
Image I

gradient computation

image gradient



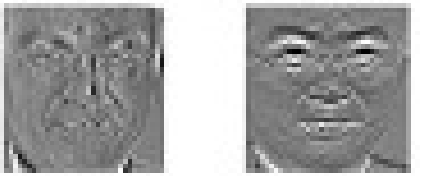
$\frac{\partial I}{\partial x}$



$\frac{\partial I}{\partial y}$

warp using the current estimate of p

warped gradient



∇I

Lucas-Kanade Registration Algorithm

algorithm

Step 4 (Jacobian):

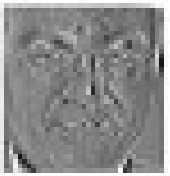
$$\frac{\partial W}{\partial p} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}$$

It represents all the partial derivatives of the warp function W with respect to the parameters vector p .

Step 5 (steepest descent images):

warped gradient

$$\nabla I \frac{\partial W}{\partial p} =$$



∇I

$$\cdot \frac{\partial W}{\partial p} =$$

the steepest descent images act as weights

Lucas-Kanade Registration Algorithm

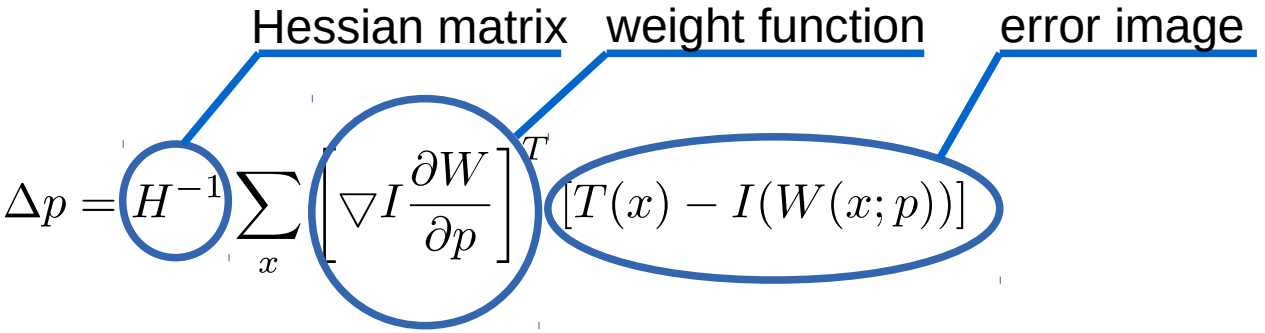
algorithm

Step 6 (Hessian matrix):

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

which corresponds to the second derivative of the Jacobian $\frac{\partial W}{\partial p}$ or also to the Jacobian of the gradient of the warp function W.

Step 7-8 (compute Δp):



Step 9 (update the parameters):

$$p = p + \Delta p$$

iterate until $||\Delta p|| < \varepsilon$

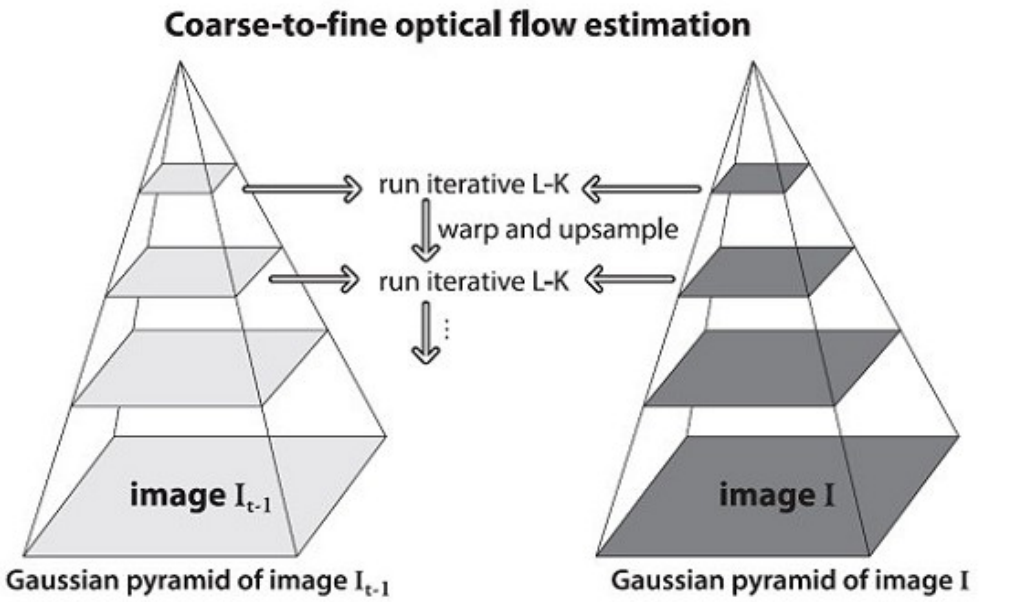
Lucas – Kanade Registration Algorithm

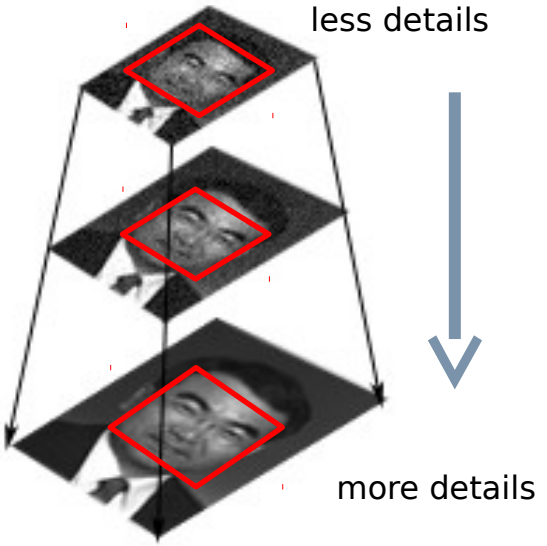
pyramidal implementation

In order to have better performance, a pyramidal approach can be used:

First the algorithm estimate the motion vector at the highest level (less details), then the obtained values are used as starting point for the next level (more details).

The height of the pyramid usually does not exceed 2-3 levels:

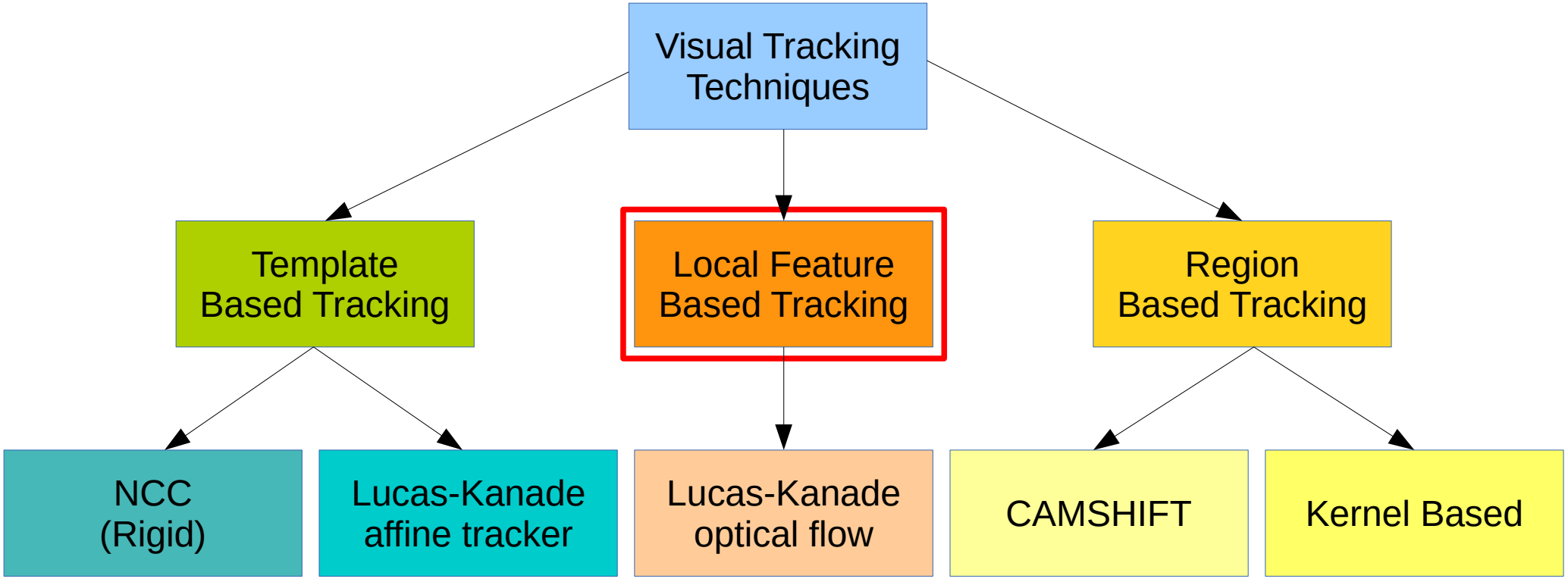




DEMO

Template Matching

Visual Tracking Techniques

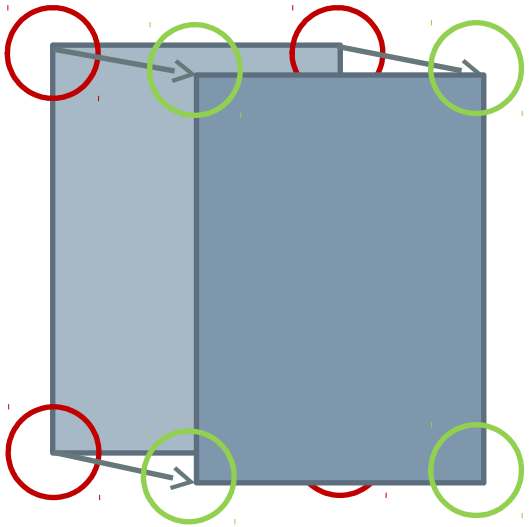


Local Feature Based Tracking

Objects are **represented** as a set of feature points (key-points or interest points) and **located** by estimating the key-points locations over time.

Local Feature Based Tracking

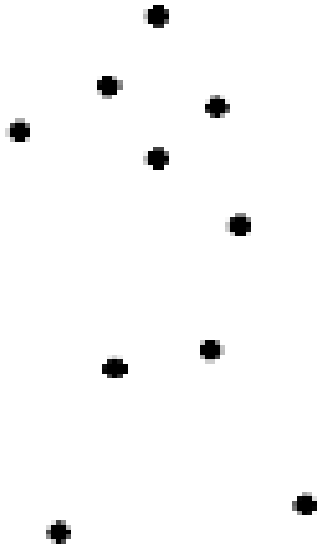
The points are tracked independently by estimating their motion vectors at each frame.



This approach is referred to as **sparse optical flow**

Optical Flow

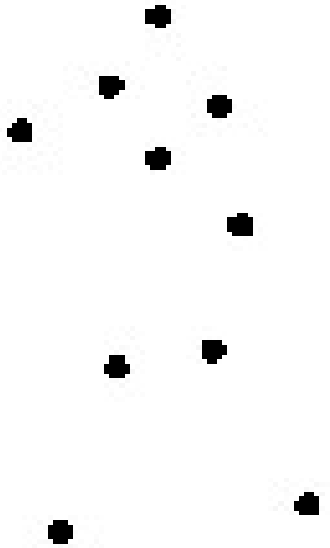
Even “impoverished” motion data can evoke a strong percept



G. Johansson, “Visual Perception of Biological Motion and a Model For Its Analysis", *Perception and Psychophysics 14, 201-211, 1973.*

Optical Flow

Even “impoverished” motion data can evoke a strong percept



G. Johansson, “Visual Perception of Biological Motion and a Model For Its Analysis", *Perception and Psychophysics 14, 201-211, 1973.*

Optical Flow

definition

Optical Flow is the pattern of apparent motion of objects, surfaces and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene.

Two types of motion:

- Movement in the scene with a static camera
- Movement of the camera

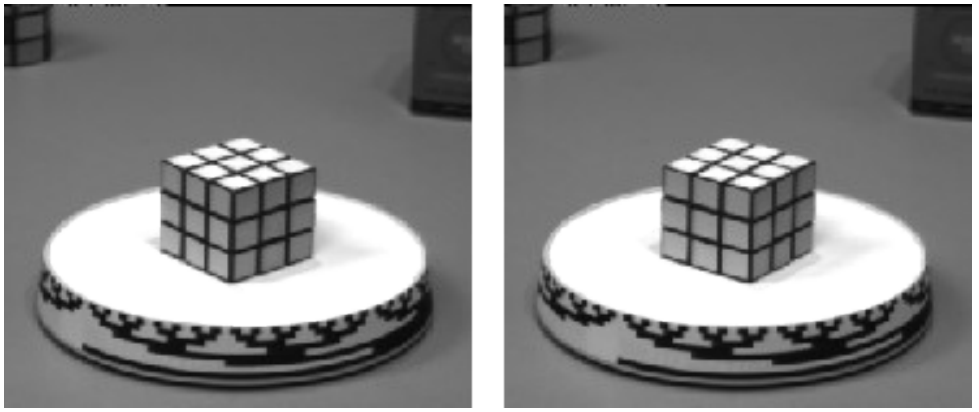
Since motion is relative anyway, these types of motion should be the same.

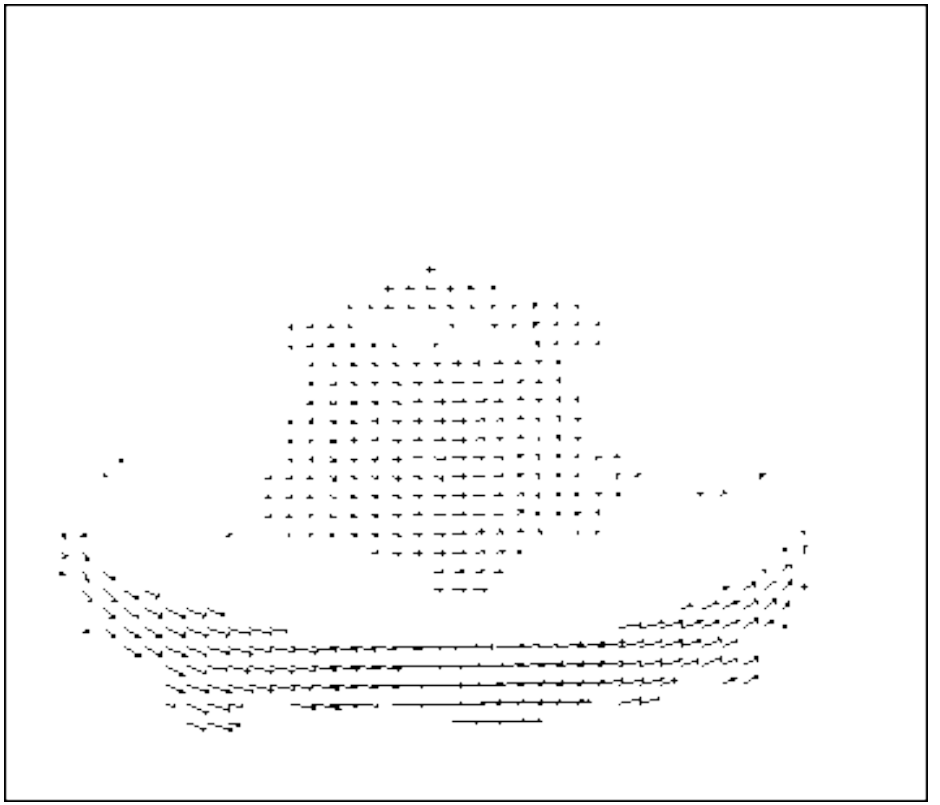
However, this is not always the case, since if the scene moves relative to the illumination, shadow effects need to be dealt with.

Optical Flow

motion field

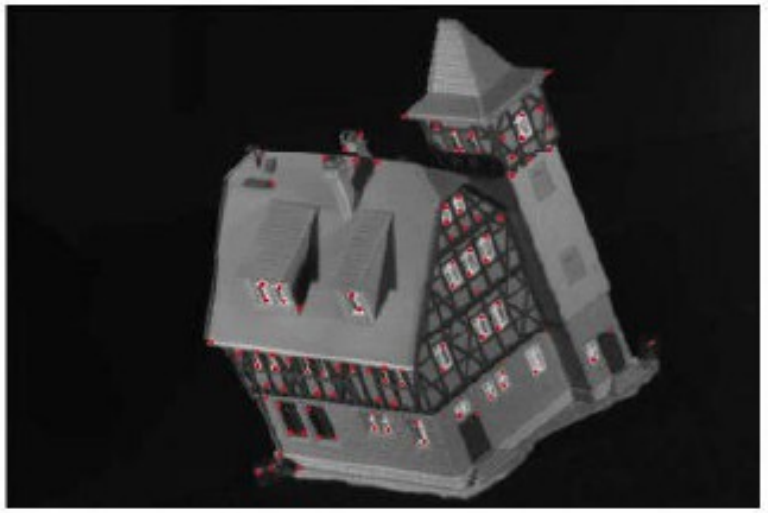
The motion field is the projection of the 3D scene motion into the image

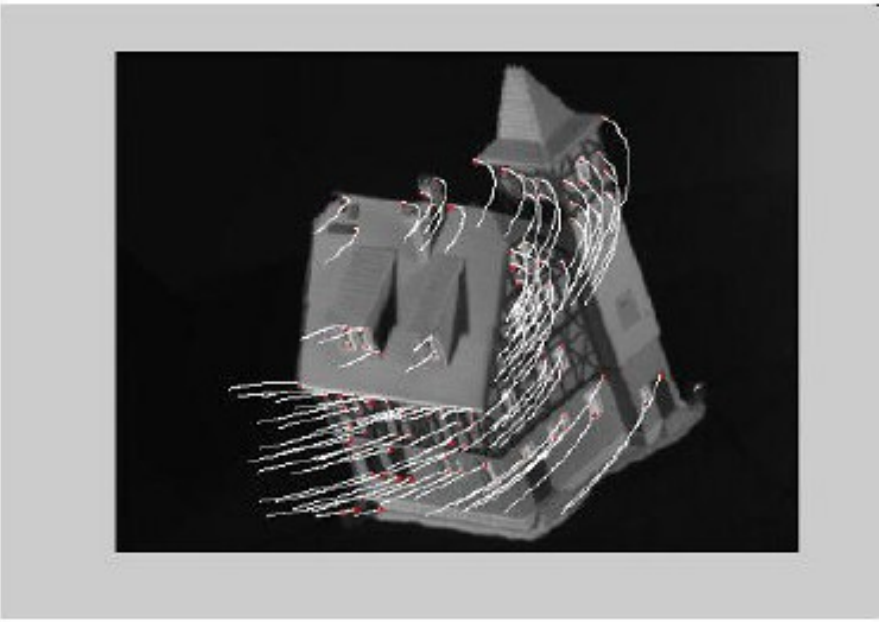




Optical Flow

motion field





Optical Flow

usages

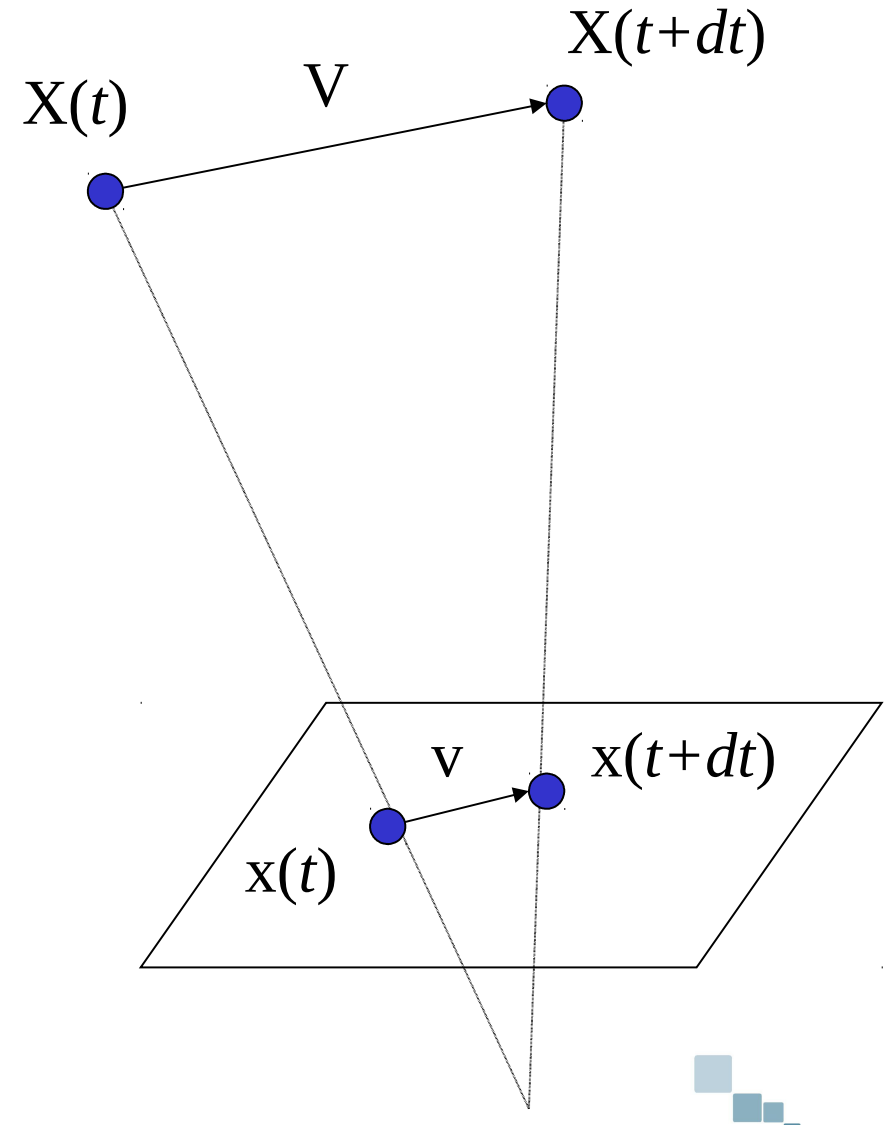
Motion flow is useful for:

- depth map,
- observer motion,
- image segmentation,
- object motion:
  - establish relationship between object & viewer,
  - guides smooth pursuit eye movement (saccades).

Optical Flow

motion field and pallax

- $\mathbf{X}(t)$ is a moving 3D point
- Velocity of scene point: $\mathbf{V} = d\mathbf{X}/dt$
- $\mathbf{x}(t) = (x(t), y(t))$ is the projection of $\mathbf{X}$ in the image
- Apparent velocity $\mathbf{v}$ in the image: given by components $vx = dx/dt$ and $vy = dy/dt$
- These components are known as the *motion field* of the image



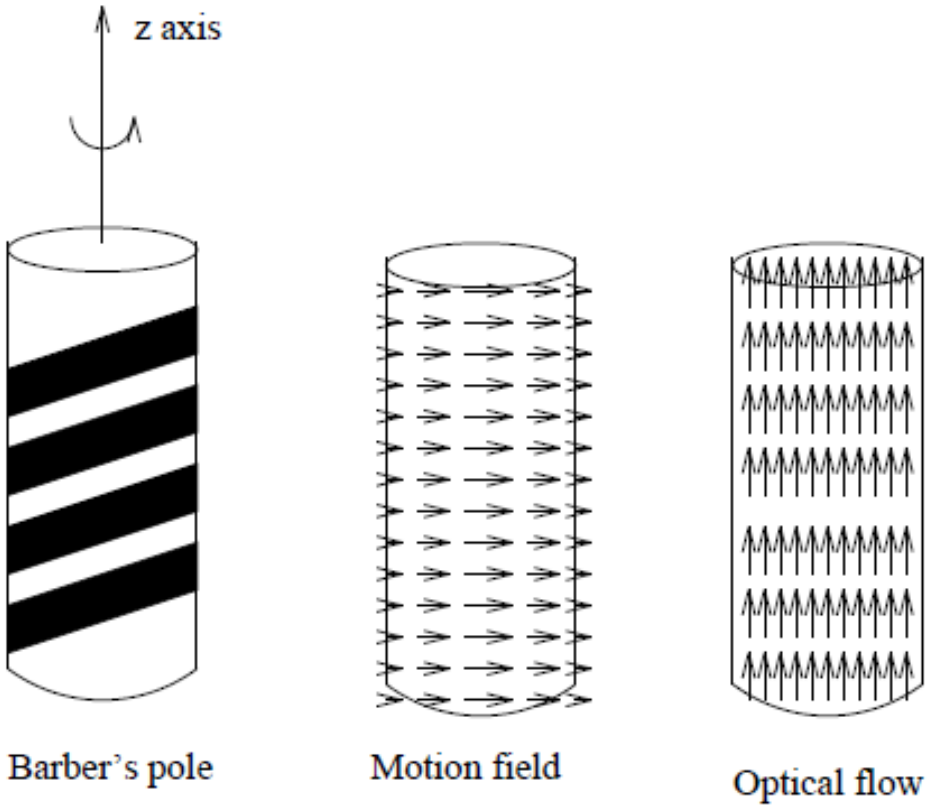
Optical Flow

aperture problem

Generally, optical flow corresponds to the motion field, but not always.

For example, the motion field and the optical flow of a rotating barber's pole are different.

In general, such cases are unusual.

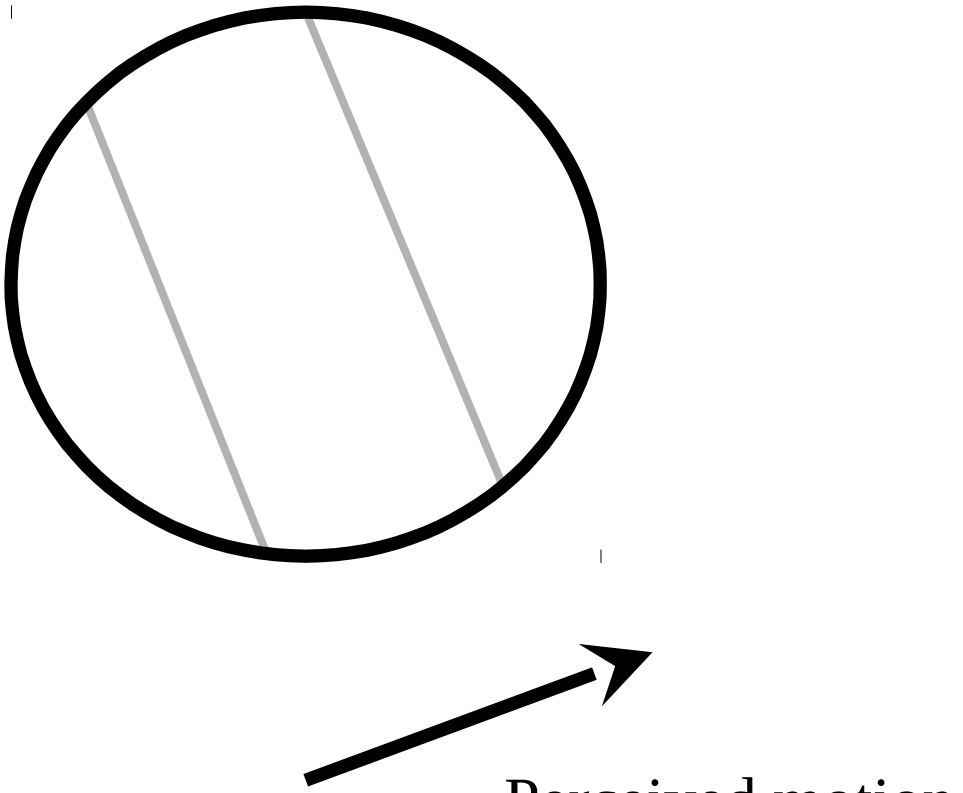


Optical Flow

aperture problem

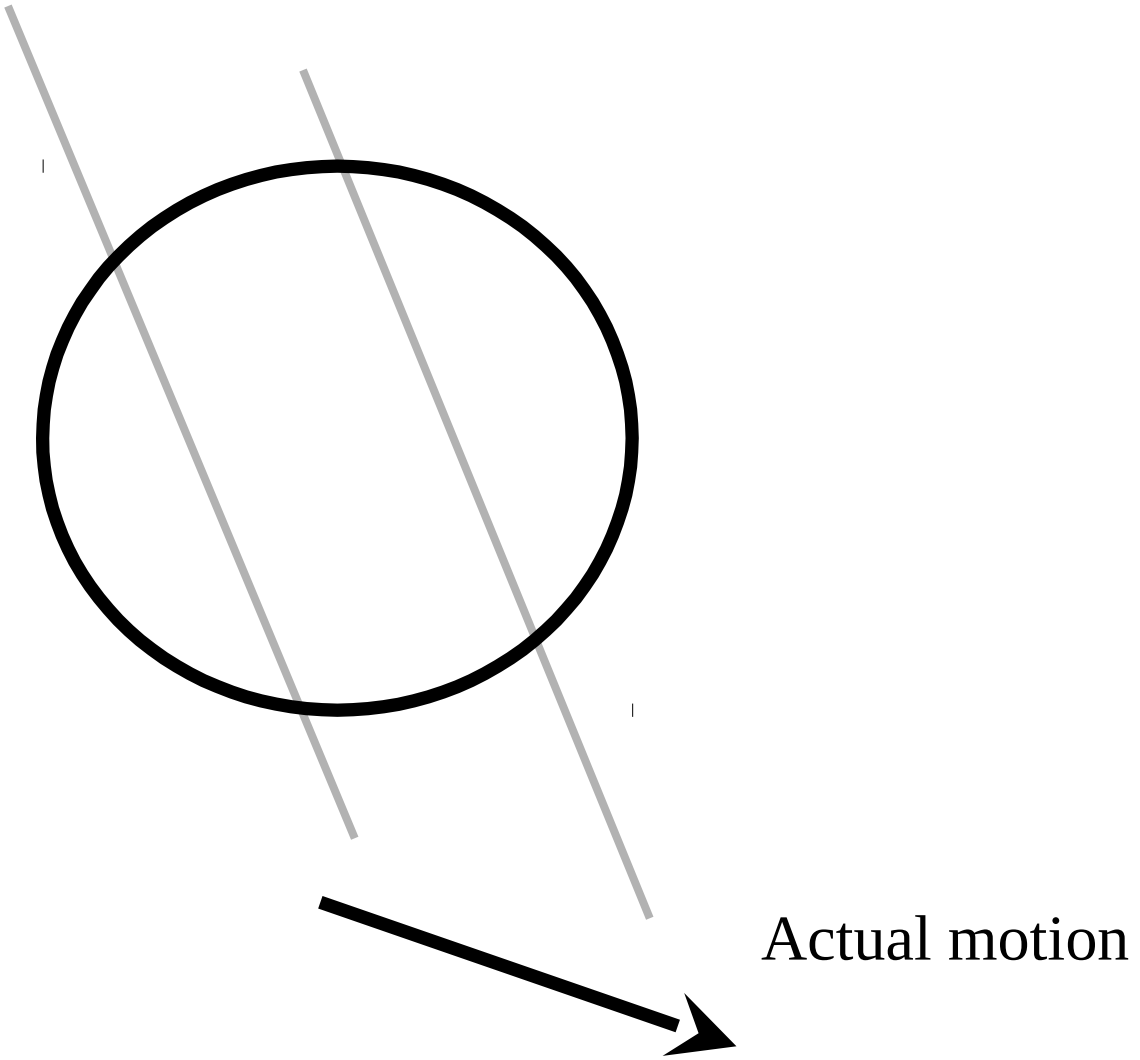
Optical Flow

aperture problem



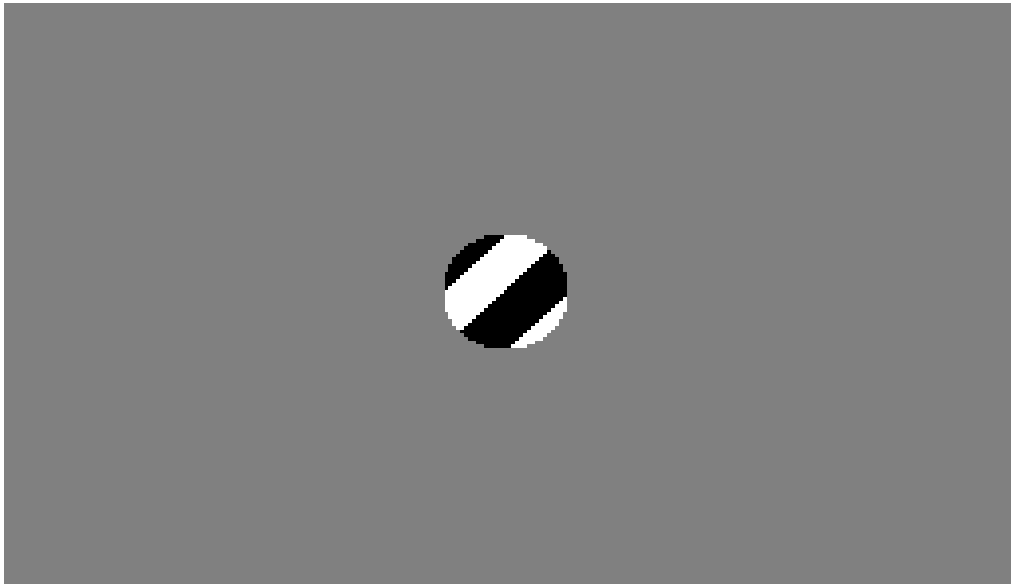
Optical Flow

aperture problem



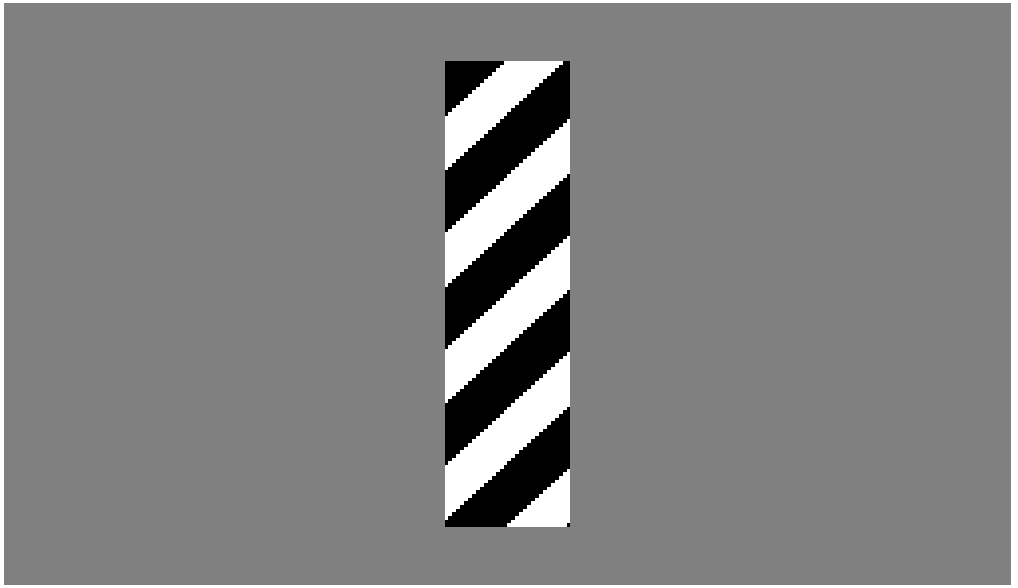
Optical Flow

aperture problem



Optical Flow

aperture problem



Optical Flow

aperture problem

Each neuron in the visual system is sensitive to visual input in a **small part** of the **visual field**, as if each neuron is looking at the visual field through a small window or **aperture**.

The motion direction of a contour is ambiguous, because the motion component **parallel** to the line cannot be inferred based on the visual input.

This means that a variety of contours of different orientations moving at different speeds can cause **identical responses** in a motion sensitive neuron in the visual system.

Optical Flow

formulation

In order to track a key-point, the **brightness constancy assumption** is usually considered. It states that the key-points can change their position but they keep approximately the same pixel value in two subsequent frames. This means that changes in image intensity are about entirely due to motion and not to lighting changes in the scene.

Given two subsequent frames respectively taken at times t and $t + \Delta t$ in order to compute the (x, y) key-point **motion vector** $(\Delta x, \Delta y)$, the **image constraint equation** is formulated using the brightness constancy assumption:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{1}$$

Expanding the right hand side with Taylor series:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + h.o.t. \tag{2}$$

Using equation (1) we get:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \Rightarrow \frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0 \tag{3}$$

Optical Flow

formulation

So we get:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{4}$$

Where V_x and V_y are the components of the velocity of the point with coordinates, (x, y) while $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the partial derivatives of I with respect to x, y and t . So we have:

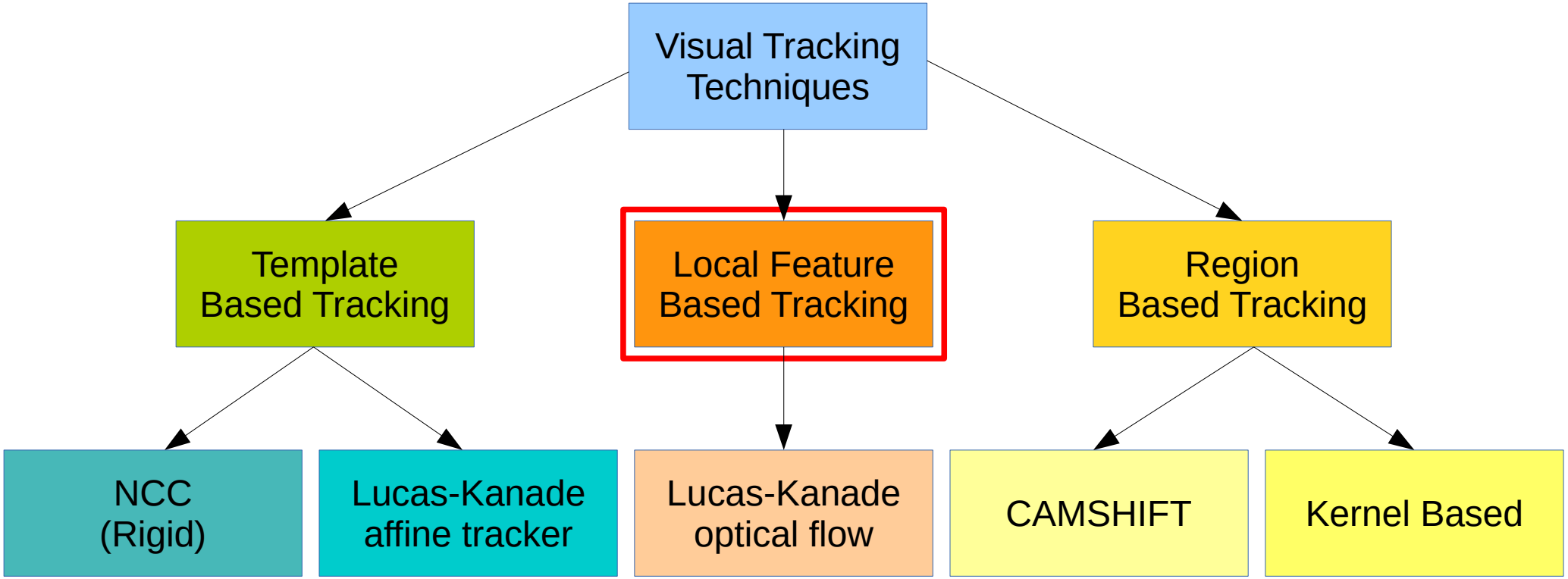
$$\left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \cdot (V_x, V_y) = -\frac{\partial I}{\partial t} \tag{5}$$

which is the **motion constraint equation**.

Considering that the spatial and temporal derivatives can be computed, the Equation (5) has two unknowns (V_x and V_y) and can not be solved. So, in order to find the optical flow, additional equation have to be added. This is done by making some additional assumption.

Each optical flow algorithm add its own assumption (i.e., equations).

Visual Tracking Techniques



Lucas-Kanade Optical Flow

assumptions

- **Brightness Constancy**: the key-points can change their position but they keep approximately the same pixel value in two subsequent frames.
- **Spatial Coherence**: neighborhood points belong to the same surface and so they share the same velocity
- **Temporal Persistence**: the movements of an object in the scene are very slow. This means that the temporal increments are faster than object movements

assumption 2 allows us to apply the L-K algorithm to an image patch surrounding the key-point

Lucas-Kanade Optical Flow

formulation

The motion constraint equation (5) can be written as:

$$I_x V_x + I_y V_y = -I_t \tag{6}$$

where $I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}, I_t = \frac{\partial I}{\partial t}$

Given the key-point (x, y), a window containing n neighbor pixels $q_1, q_2, \ldots, q_n$ is considered and the following equations are obtained from (6):

$$\begin{array}{rcl} I_x(q_1)V_x + I_y(q_1) + V_y & = & -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2) + V_y & = & -I_t(q_2) \\ \ldots & & \\ I_x(q_n)V_x + I_y(q_n) + V_y & = & -I_t(q_n) \end{array}$$

We can write the system as:

$$Av = b \tag{7}$$

Lucas-Kanade Optical Flow

formulation

where:

$$\mathrm{A}=\begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \dots & \\ I_x(q_n) & I_y(q_n) \end{bmatrix}; v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \text{and} \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \dots \\ -I_t(q_n) \end{bmatrix}$$

The equation (7) represents an over-determined system which can be solved with the least squares as follows:

$$A^T A v = A^T b \Rightarrow v = (A^T A)^{-1} A^T b$$

The optical flow (i.e., the motion vector) is finally computed as:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i^n I_x(q_i)^2 & \sum_i^n I_x(q_i) I_y(q_i) \\ \sum_i^n I_y(q_i) I_x(q_i) & \sum_i^n I_y(q_i)^2 \end{bmatrix} \begin{bmatrix} -\sum_i^n I_x(q_i) I_t(q_i) \\ -\sum_i^n I_y(q_i) I_t(q_i) \end{bmatrix}$$

structure tensor

Lucas-Kanade Optical Flow

good features to track

Not all parts of an image contain motion information. Similarly, along a straight edge, we can only determine the motion component orthogonal to the edge.

In order to overcame this problem, generally the feature point is chosen in a **high texture** area of the image (e.g., a corner) avoiding to select it from a **uniform area**.

Looking at the equation (7) we can retrieve some information about the selected feature. Specifically the 2×2 coefficients matrix A must be both above the image noise and *well-conditioned*.

Let λ_1 and λ_2 be the eigenvalues of A. Two small values would determine an almost constant intensity in the selected patch (a uniform area), white two large eigenvalues can represent **corners** or **high texture** areas.

In practice when the eigenvalues are **large enough** to avoid the noise, the corresponding matrix is well-conditioned. So we impose:

$$min(\lambda_1, \lambda_2) > \lambda$$

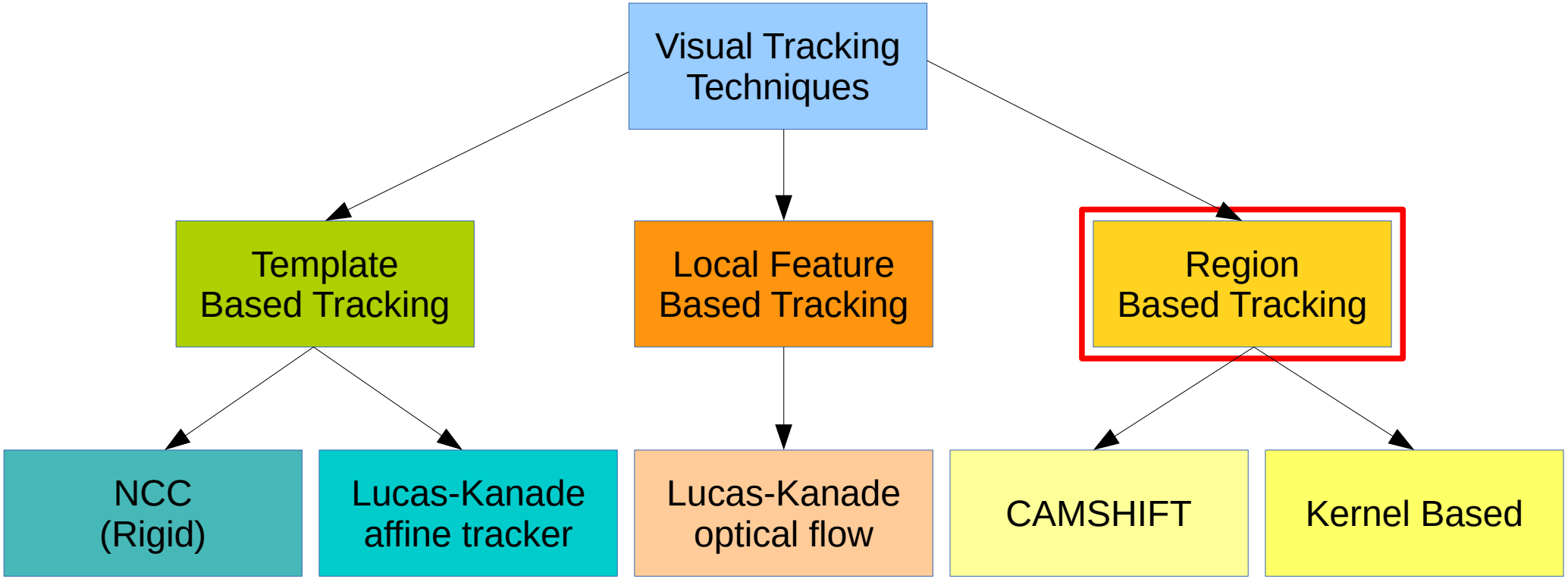
where λ is a predefined threshold.

like in the Harris corner detector

DEMO

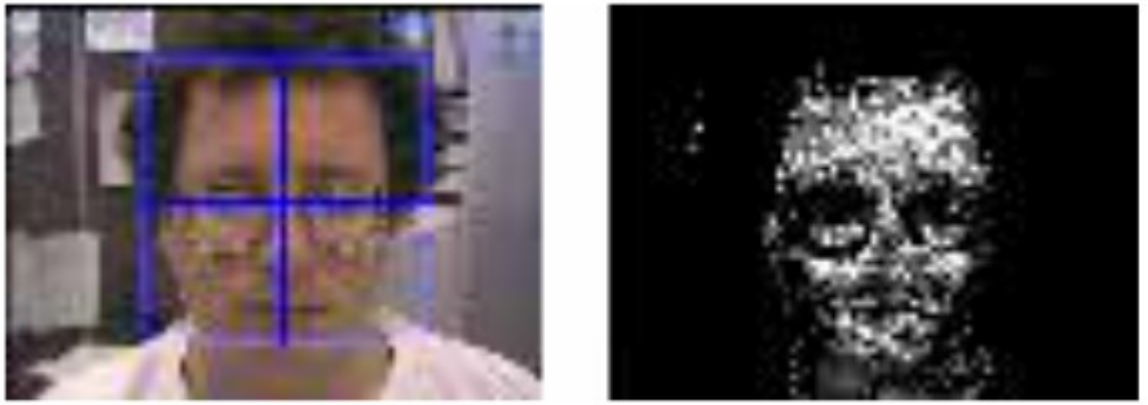
Lucas-Kanade Optical Flow

Visual Tracking Techniques



Region Based Tracking

Objects are **represented** as probability distributions describing their region of appearance in some *feature space* and **located** by finding the region whose probability distribution is the closest to the object one according to some similarity measure.



feature space: hue colors

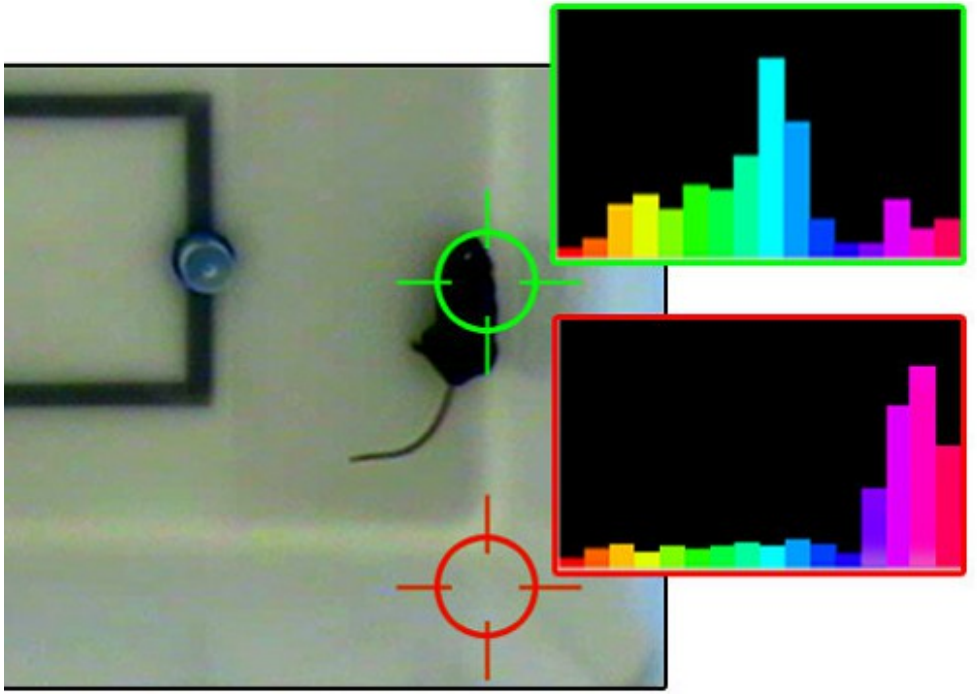
Region Based Tracking

feature space

The feature space must be chosen in order to maximize the separation between the **object description** and the **background description**.

In this case the object is represented as a **color distribution** (histogram).
The selected feature space is such that the object **representation** (color distribution) is **different** enough from the background one.

Many possible feature spaces:
- Color
- Intensity (grayscale)
- Edge orientations
- ...



The MeanShift Procedure

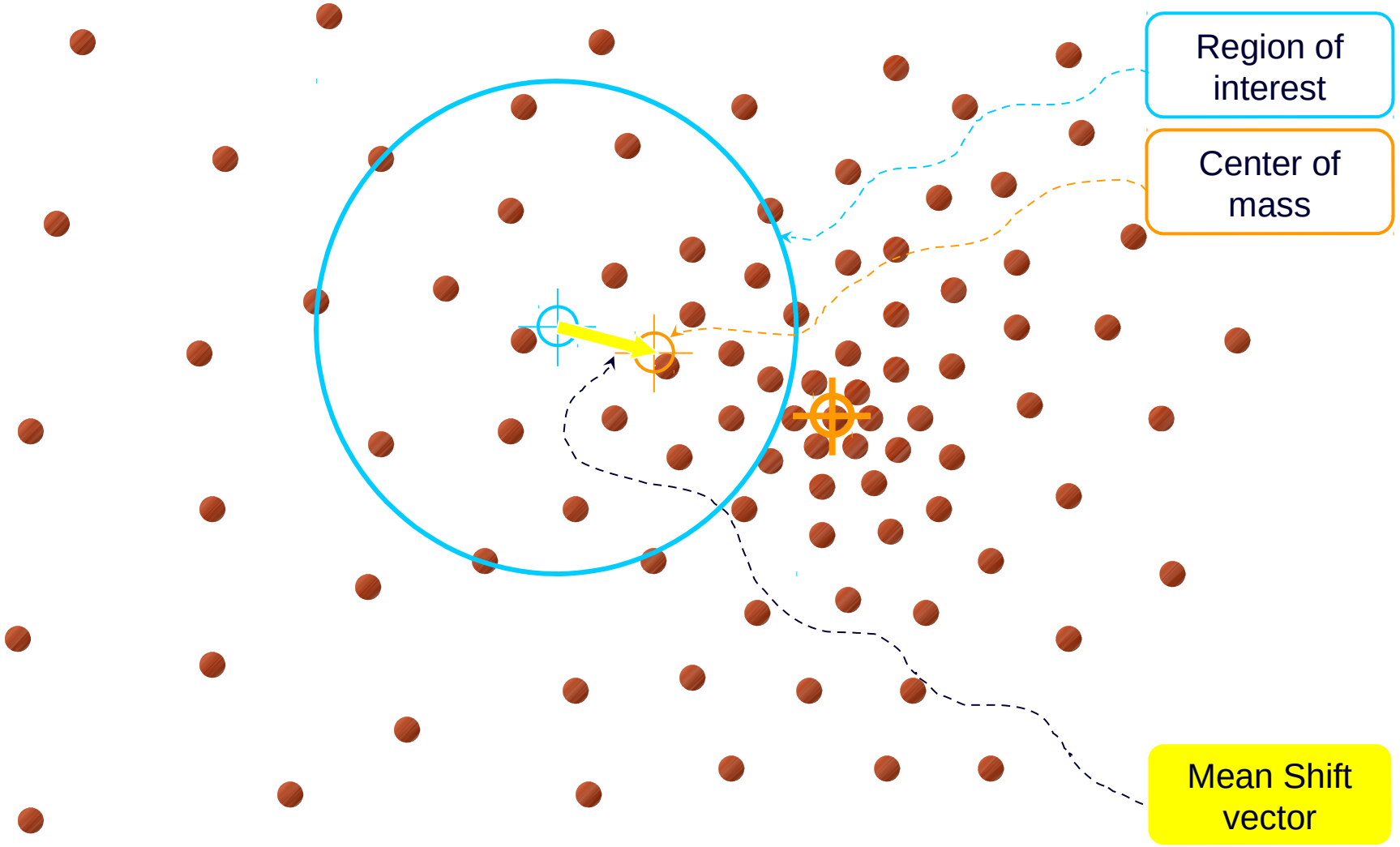
Mean shift is an iterative procedure for locating the maxima of a density function (the distribution mode) given discrete data sampled from that function.

It is a *hill climbing* technique which locally searches for the best path to reach the mode (the hilltop) of the distribution.

Algorithm description:

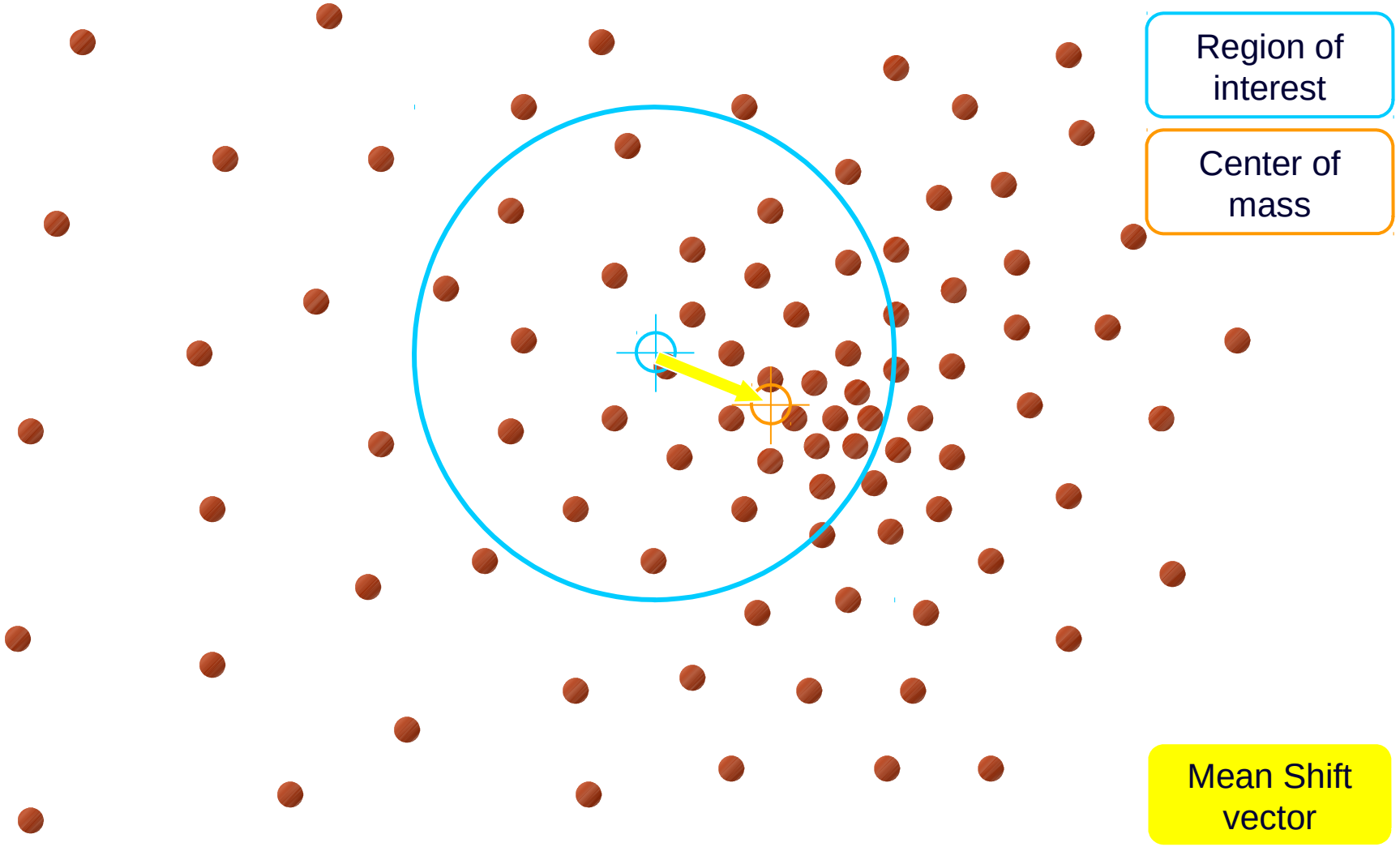
1. Choose a search window size
2. Choose the initial location of the search window
3. Compute the mean location in the search window
4. Center the search window at the mean location computed in step 3
5. Repeat steps 3 and 4 until convergence (until the mean location moves less that a fixed threshold)

Gradient-based optimization: Intuitive description



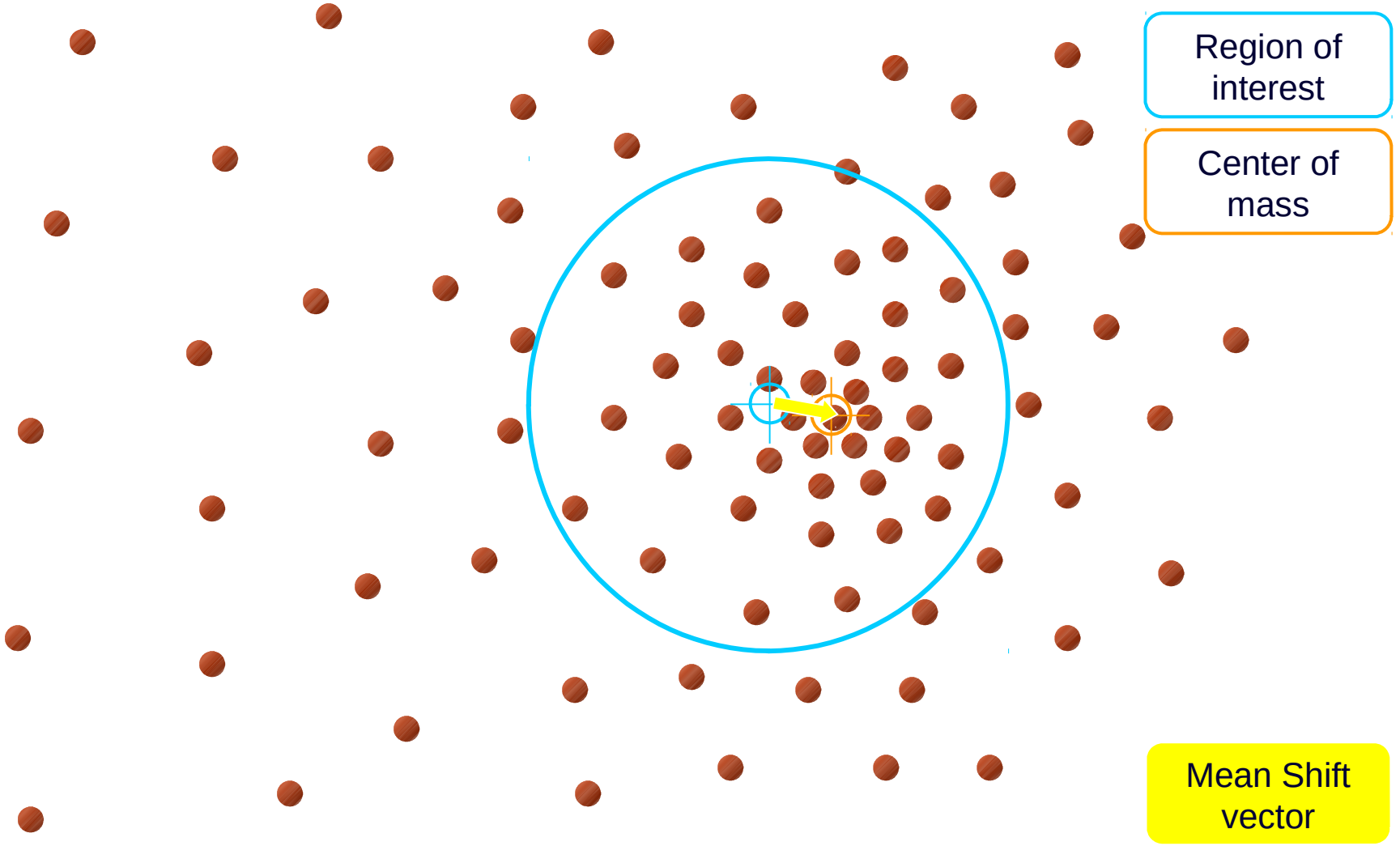
<u>Objective</u> : Find the densest region

Gradient-based optimization: Intuitive description



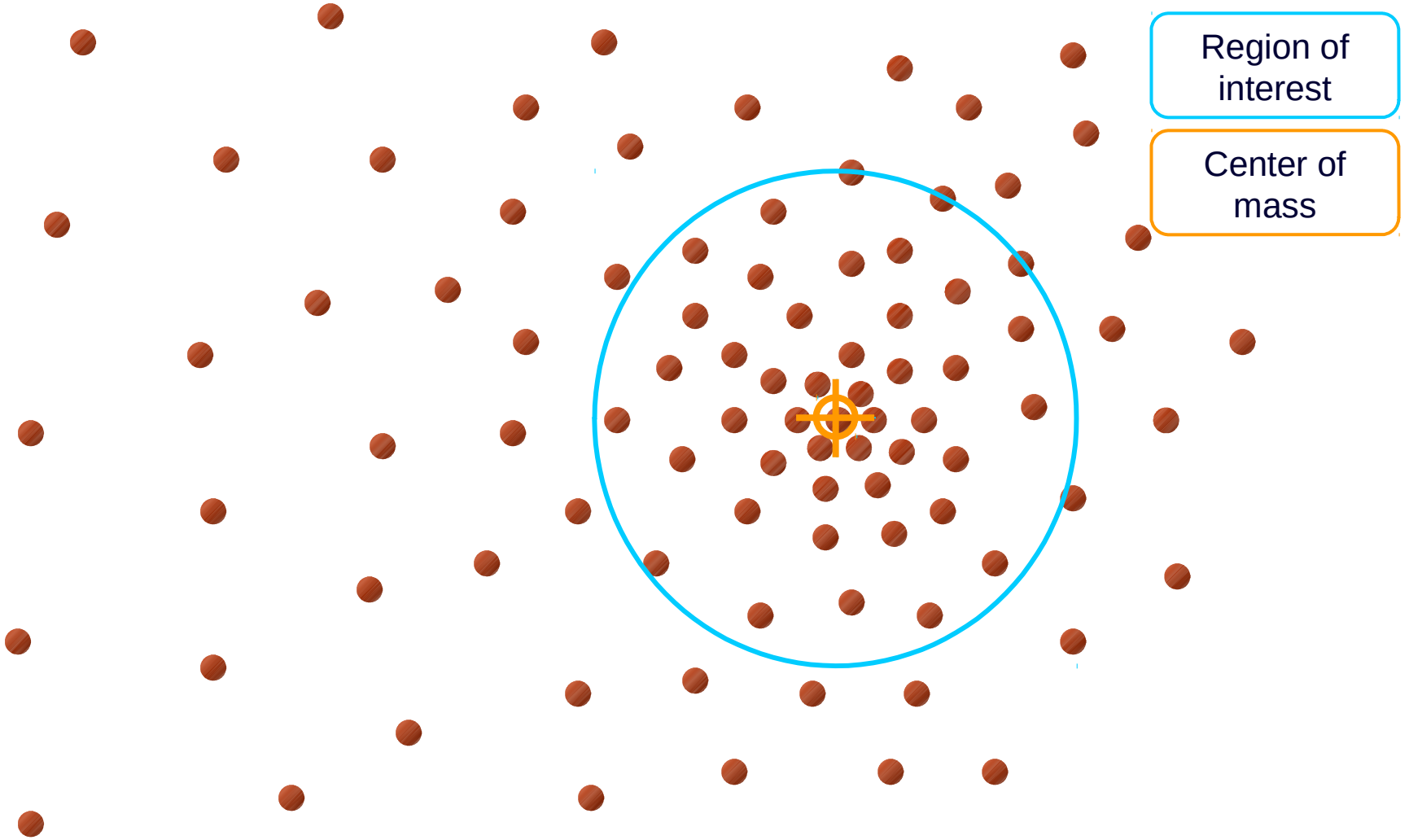
<u>Objective</u> : Find the densest region

Gradient-based optimization: Intuitive description



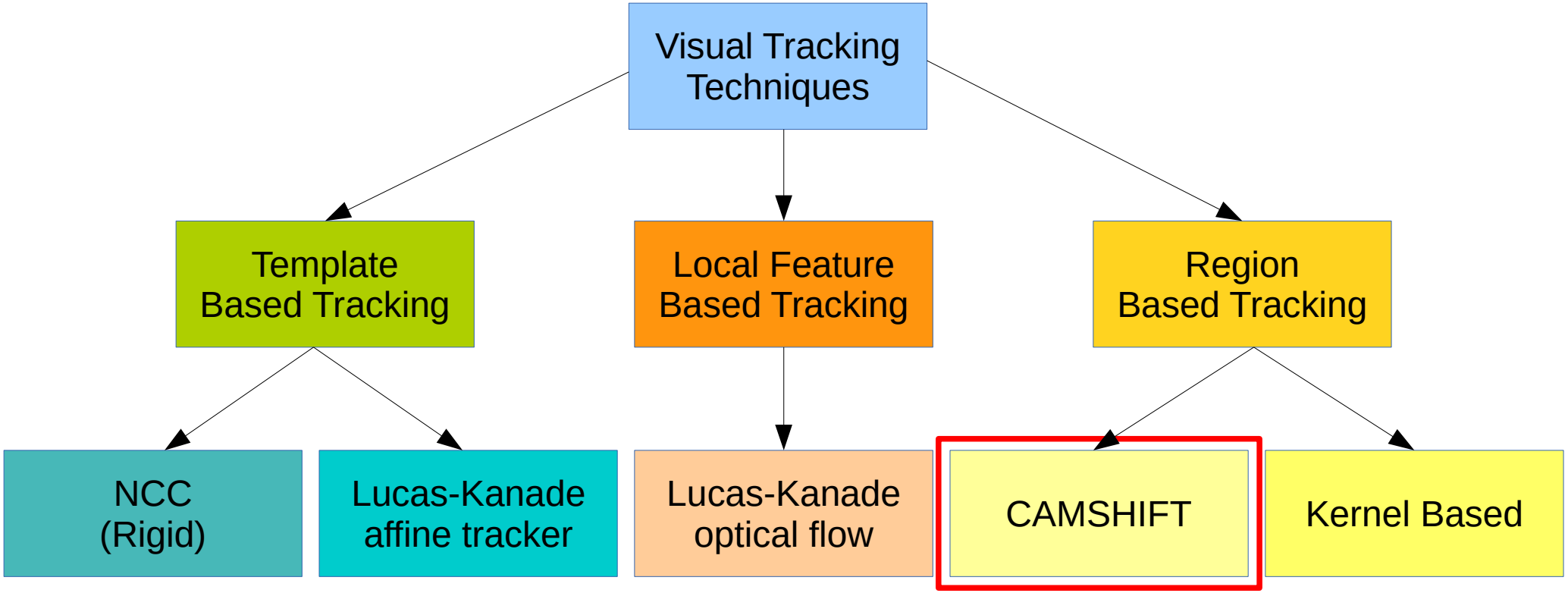
<u>Objective</u> : Find the densest region

Gradient-based optimization: Intuitive description



<u>Objective</u> : Find the densest region

Visual Tracking Techniques



CAMSHIFT

definitions

The base idea behind the CAMSHIFT algorithm is to use the MeanShift procedure in order to **find the mode** of a probability image representing the probability to find the object at each frame position.

In order to build a **probability image** out of the current frame, the target object is first represented in the chosen feature space as a quantized probability density function.

Let

$$q = \{q_u\}_{u=1\ldots m} \sum_{u=1}^{m} q_u = 1$$

be the **quantized probability function** representing the object and let the function

$$b : \mathbb{R}^2 \rightarrow \{1 \ldots m\}$$

associate to the pixel (x, y) the index of its bin in the quantized feature space.

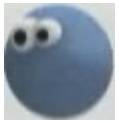
The **probability image** is defined as:

$$I(x, y) = q_u | u = b(x, y)$$

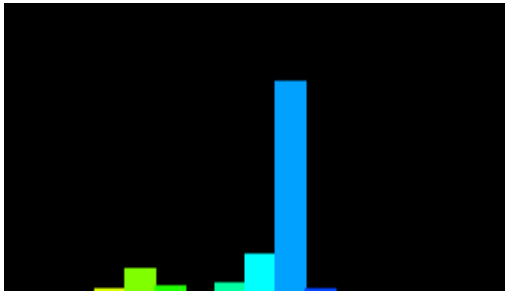
CAMSHIFT

algorithm illustration

target object

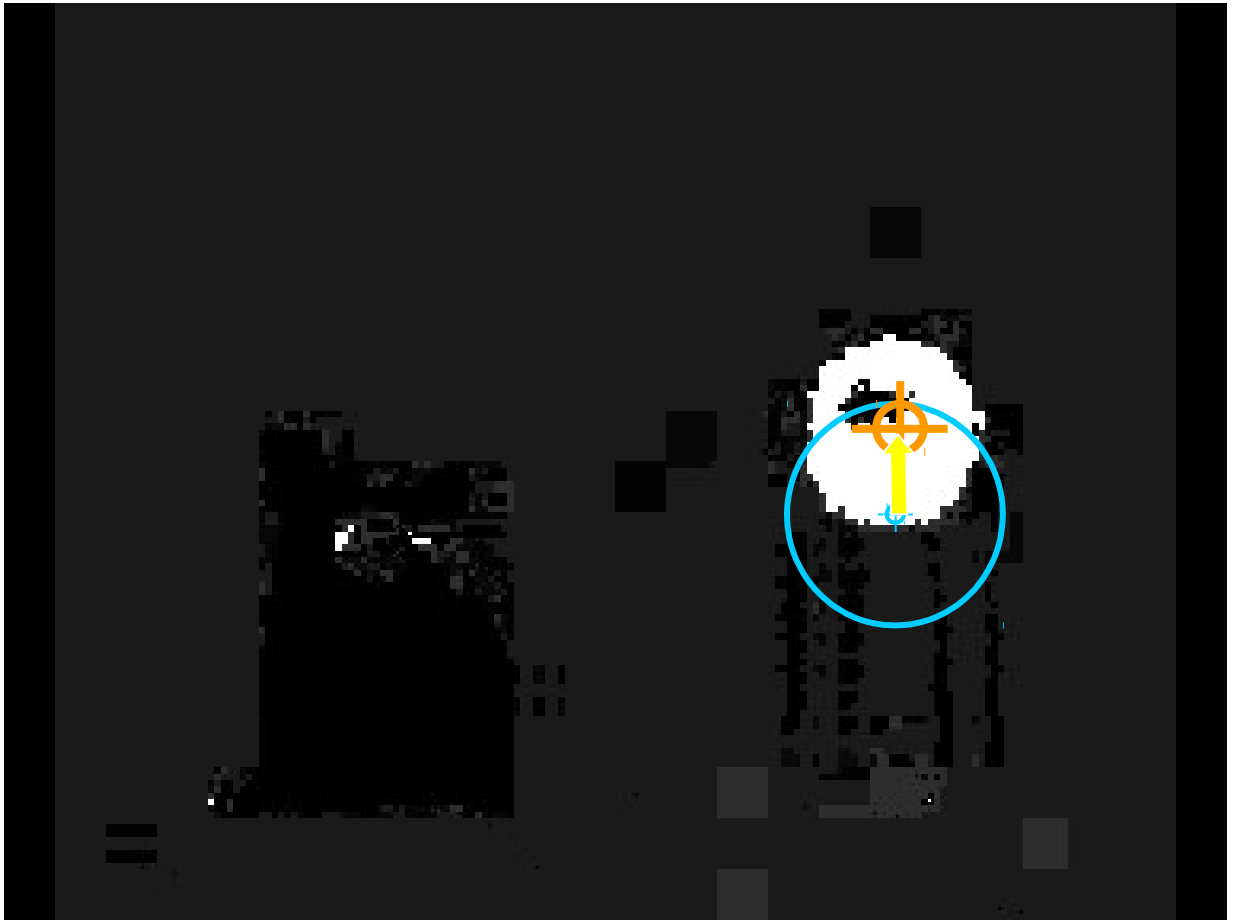


template model



input frame

probability image



CAMSHIFT ← Continuously Adaptive Mean Shift

dealing with scale

Since the probability **distributions** derived from video sequences are **not static** (due to the object change of location and scale, and **adaptive** version of the MeanShift procedure is used.

Specifically the MeanShift search **window scale** is **updated** during the search process. The **localization process** (both the search and the window size update) are performed calculating the following quantities at **each step**:

$$M_{00} = \sum_x \sum_y I(x,y) \quad \text{(zeroth moment)} \qquad M_{10} = \sum_x \sum_y xI(x,y);\ M_{01} = \sum_x \sum_y yI(x,y)$$

where I is the probability image while x and y range over the current search window.

The new location is computed as:

$$x_c = \frac{M_{10}}{M_{00}};\ y_c = \frac{M_{01}}{M_{00}}$$

while the **window size** is set to a **function of the zeroth moment**.

the zeroth moment is the distribution "area" found under the search window

CAMSHIFT

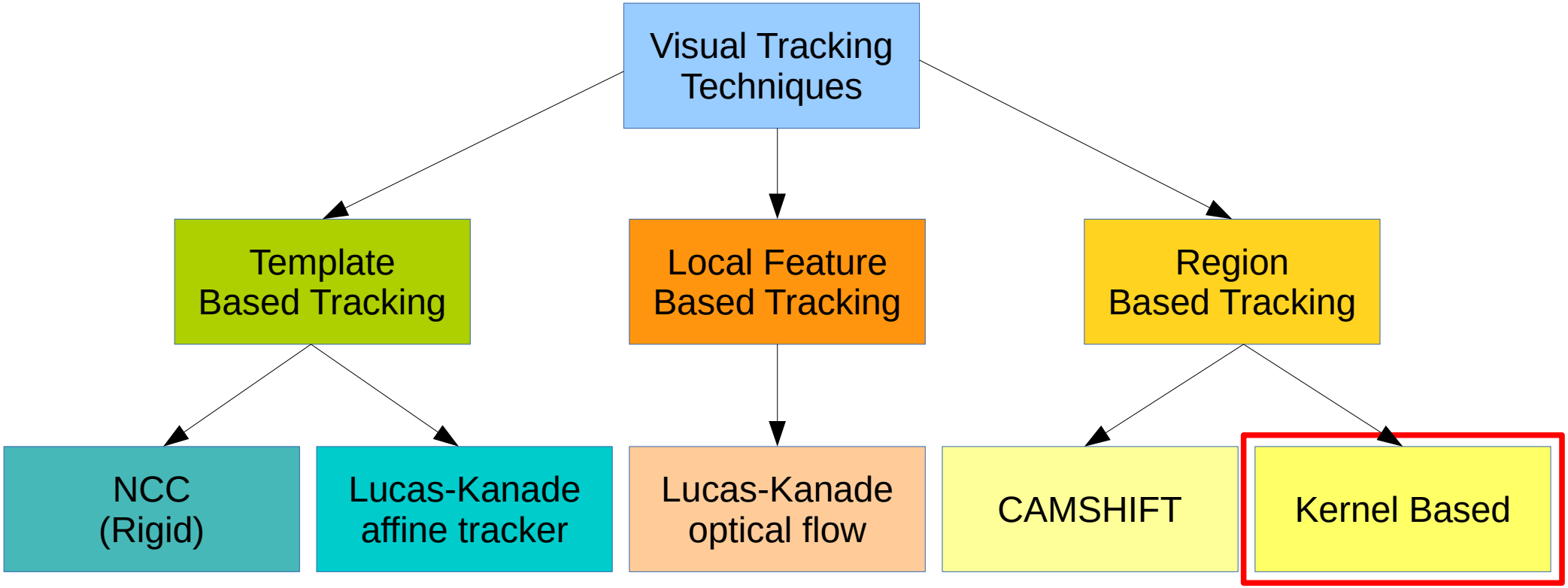
summary

1. Chose the initial size and location of the search window
2. Compute the zeroth moment $M_{00} = \sum_x \sum_y I(x,y)$ and the moments $M_{10} = \sum_x \sum_y xI(x,y);\ M_{01} = \sum_x \sum_y yI(x,y)$ within the current search window
3. Set the new window position to $x_c = \frac{M_{10}}{M_{00}};\ y_c = \frac{M_{01}}{M_{00}}$ (MeanShift procedure) and update the window size to a function of the zeroth moment
4. Repeat steps 2 and 3 until convergence (until $|(x,y)| < \varepsilon$)

DEMO

CAMSHIFT

Visual Tracking Techniques



Kernel Based Object Tracking

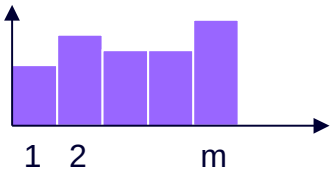
The base idea is to use the MeanShift procedure to find the **nearest peak** of the similarity measure between the target area representation (probability distribution) and the candidate area one.

In order to **regularize** the similarity function an **isotropic kernel** in the spatial domain is used in order to obtain a **smooth similarity function** (suitable for gradient-based optimizations).

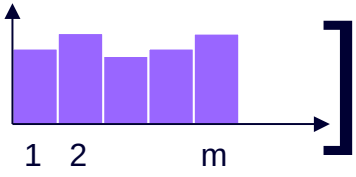
The representation of the region containing the object is called **target model**, whereas the representation of a region in the frame where the object is searched is called **target candidate**.

Kernel Based Object Tracking Overview

Requirements:
Real-time
Robustness

$\rho[$ 

target model

, 

target candidate
New target model

$]$

1) Detect the object (target model) in the current frame

2) Represent the target model by its PDF in the feature space (e.g. quantized color space)

3) Start from the position of the target model in the next frame

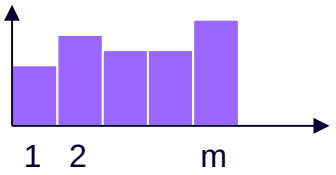
4) Search in the neighborhood the best target candidate by maximizing a similarity function

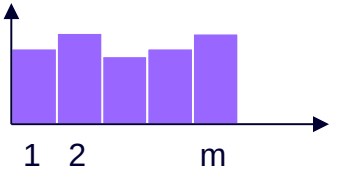
5) Repeat the same process in the next pair of frames

Gradient-Based Optimization

Target Model
(centered at 0)

Target Candidate
(centered at y)

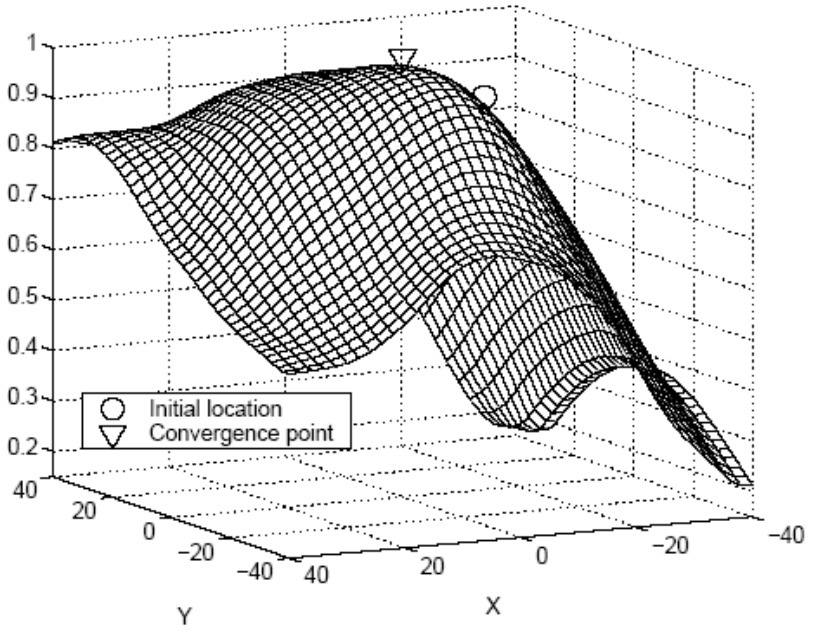




$$\vec{q} = \{q_u\}_{u=1\ldots m} \sum_{u=1}^{m} q_u = 1$$

$$\vec{p}(y) = \{p_u(y)\}_{u=1\ldots m} \sum_{u=1}^{m} p_u = 1$$

$\hat{\rho}(\mathbf{y}) \equiv \rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}]$ Similarity Function



The similarity function plays the role of a likelihood and its **local maxima in the image indicate the presence of objects in the second frame having representations similar to the target model defined in the first frame**.

If **only spectral information** is used to characterize the target, the **similarity function can have large variations for adjacent locations** on the image lattice and the spatial information is lost.

To find the maxima of such functions, **gradient-based optimization procedures are difficult to apply** and only an expensive exhaustive search can be used.

To **regularize the similarity function an isotropic kernel in the spatial domain** can be used.
When the kernel weights, carrying continuous spatial information, are used in defining the feature space representations, the similarity function becomes a smooth function.

Kernel Based Object Tracking

target representation

Chosen a feature space, the target is represented as a probability density function (pdf) q. Without loss of generality the object is considered as centered at location $(0, 0)$.

In the subsequent frame a target candidate is defined at location y and is characterized by its pdf $p(y)$. The pdfs are estimated from the data in the form of discrete densities, i.e., m-bins normalized histogram. So we have:

$$\begin{array}{rcl} \text{target model:} & \hat{q} = \{q_u\}_{u=1\ldots m} & \sum_{u=1}^{m} q_u = 1 \\ \text{target candidate:} & \hat{p}(y) = \{p_u(y)\}_{u=1\ldots m} & \sum_{u=1}^{m} p_u = 1 \end{array}$$

The similarity function between p and q is denoted by:

$$\rho(y) \equiv \rho[\hat{p}(y), \hat{q}]$$

The function $\rho(y)$ plays the role of a likelihood and its local maxima represent the presence of an object in the frame having representation similar to q defined in the first frame.

This kind of similarity function can have large variations for adjacent positions, making gradient optimizations (i.e., MeanShift) difficult to apply.

The similarity function is regularized (smoothed) by masking the objects with an isotropic kernel in the spatial domain.

Kernel Based Object Tracking

isotropic kernel

A target is represented as an ellipsoidal region in the image. To eliminate the influence of different target dimensions, each target is first normalized to the unit circle.

Target Model

$$\hat{q}_u = C \sum_{i=1}^{n} k(||x_i^*||^2)\delta[b(x_i^*) - u] \quad (1)$$

$$C = \frac{1}{\sum_{i=1}^{n} k(||x_i^*||^2)}$$

Target Candidate

$$\hat{p}_u(y) = C_h \sum_{i=1}^{n_h} k(\| \frac{y - x_i}{h} \|^2)\delta[b(x_i) - u] \quad (2)$$

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

where $\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$ is the Kronecker function and $b : \mathbb{R}^2 \rightarrow \{1 \dots m\}$ associates to the pixel x_i^* the index of its bin in the quantized feature space.

C and C_h are derived from the normalization conditions $\sum_{u=1}^{m} q_u = 1$ and $\sum_{u=1}^{m} p(y)_u = 1$.

k is the profile of the isotropic kernel used to assign smaller weights to the pixels further from the center. h is the bandwidth, a parameter related to the candidate scale.

Kernel Based Object Tracking

similarity measure

In order to compare two discrete pdfs, a **metric** derived from the Bhattacharya coefficient is used. The **distance** between two discrete distributions is defined as:

$$d(y) = \sqrt{1 - \rho[p(y), q]} \tag{3}$$

where it is chosen:

this is the function we want to maximize with respect to y

$$\rho(y) \equiv \rho[p(y), q] = \sum_{u=1}^{m} \sqrt{p_u(y) q_u} \tag{4}$$

similarity function smoothness

The similarity function (4) inherits the properties of the kernel profile $k(x)$ when the target model and the target candidate are represented according to (1) and (2). A differentiable kernel profile yields a differentiable similarity function and efficient gradient-based optimizations procedures can be used to find the maxima (or equivalently the minima of (3)).

The presence of the continuous kernel introduces an interpolation process between the locations of the image lattice.

Kernel Based Object Tracking

distance minimization

Minimizing (3) is equivalent to maximizing (4). Using Taylor expansions around the values $\hat{p}_u(\hat{y}_0)$ the linear approximation of the Bhattacharyya coefficient (4) is obtained after some manipulations as:

$$\rho[\hat{p}(y), \hat{q}] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{\hat{p}_u(\hat{y}_0)} + \frac{C_h}{2}\sum_{i=1}^{n_h} w_i k\left(\left\|\frac{y - x_i}{h}\right\|^2\right) \tag{5}$$

where

$$w_i = \sum_{u=1}^{m}\sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}}\delta[b(x_i) - u] \tag{6}$$

which is verified when the target candidate $\{\hat{p}_u(y)\}_{u=1\ldots m}$ does not change drastically from the initial $\{\hat{p}_u(y_0)\}_{u=1\ldots m}$, which is most often a valid assumption between consecutive frames.

The condition $\hat{p}_u(\hat{y}_0) > 0$ can be enforced by not using the feature values in violation.

The MeanShift procedure is so applied moving iteratively the current location $\hat{y}_0$ to the new location $\hat{y}_1$ according to the relation:

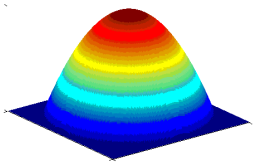
$$y_1 = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\|\frac{\hat{y}_0 - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{\hat{y}_0 - x_i}{h}\right\|^2\right)} \tag{7}$$

where $g(x) = -k'(x)$

Kernel Based Object Tracking

Epanechnikov kernel

The Epanechnikov kernel profile is defined as follows:



$$k(x) = \begin{cases} \frac{1}{2} C_d^{-1} (d+2)(1 - ||x||^2) & \text{if } ||x|| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

where d is the dimension of the space which the x points belong to and C_d is the volume of the d-dimensional unit sphere. In our case: $d = 2$ and $C_d = \frac{4}{3}\pi$.

Since the derivative $g(x)$ of the Epanechnikov profile is constant, (7) reduces to:

$$y_1 = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\| \frac{\hat{y}_0 - x_i}{h} \right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\| \frac{\hat{y}_0 - x_i}{h} \right\|^2\right)} \quad (7)$$

$$y_1 = \frac{\sum_{i=1}^{n_h} x_i w_i}{\sum_{i=1} n_h w_i} \quad (8)$$

Kernel Based Object Tracking

algorithm

1. Initialize the location of the target in the current frame with y_0, compute the **histogram** $\hat{p}(y_0)$ using

$$\hat{p}_u(y) = C_h \sum_{i=1}^{n_h} k(\| \frac{y - x_i}{h} \|^2) \delta[b(x_i) - u]$$

   where $k(x)$ is the Epanechnikov kernel profile.

2. Compute the **weights** using:

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{q_u}{p_u(y_0)}} \delta[b(x_i) - u)$$

3. Find the **next location** of the target candidate using:

$$y_1 = \frac{\sum_{i=1}^{n_h} x_i w_i}{\sum_{i=1}^{n_h} w_i}$$

4. If $\|y_1 - y_0\| < \varepsilon$ stop. Otherwise set $y_0 = y_1$ and go to step 2.

Kernel Based Object Tracking

adaptive scale

During the tracking, the target object scale can change as it come closer or go further from the camera.

In order to deal with the scale change, we act on the bandwidth (h) parameter:

- $h < 1$ values determine smaller n_h values and so less pixels are taken into account when the representation of the target candidate is computed, that corresponds to actually searching for the object at a smaller scale.
- Similarly, using $h > 1$ values, correspond to searching for the object at a bigger scale

So in order to deal with scale, the object can be searched using three bandwidth values:

- $h = h_{prev} - \Delta h$
- $h = h_{prev}$
- $h = h_{prev} + \Delta h$

where h_{prev} is the previous bandwidth parameter and Δh is a predefined increment value. The bandwidth value yielding the best Bhattacharyya result is denoted by h_{opt} .

To avoid over-sensitive scale adaptation, the bandwidth associated width the current frame is obtained through filtering:

$$h_{new} = \gamma h_{opt} + (1 - \gamma) h_{prev}$$

where $\gamma = 0.1$ by default.

DEMO

MEANSHIFT

References

[1] Lucas, B. D., & Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. (Array, Ed.)Imaging, 130(x), 674–679.

[2] Baker, S., Gross, R., Ishikawa, T., & Matthews, I. (n.d.). Lucas-Kanade 20 Years On : A Unifying Framework : Part 2 1 Introduction.

[3] Tomasi, C., & Kanade, T. (1991). Detection and Tracking of Point Features Technical Report CMU-CS-91-132, (April).

[4] Comaniciu, D., Meer, P., & Member, S. (2002). Mean Shift : A Robust Approach Toward Feature Space Analysis, 24(5), 603–619.

[5] Bradski, G. R. (1998). Computer Vision Face Tracking For Use in a Perceptual User Interface. Interface, 2(2), 12–21.

[6] Comaniciu, D., Ramesh, V., & Meer, P. (2003). Kernel-based object tracking. (V. Ramesh & P. Meer, Eds.)IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(5), 564–577.