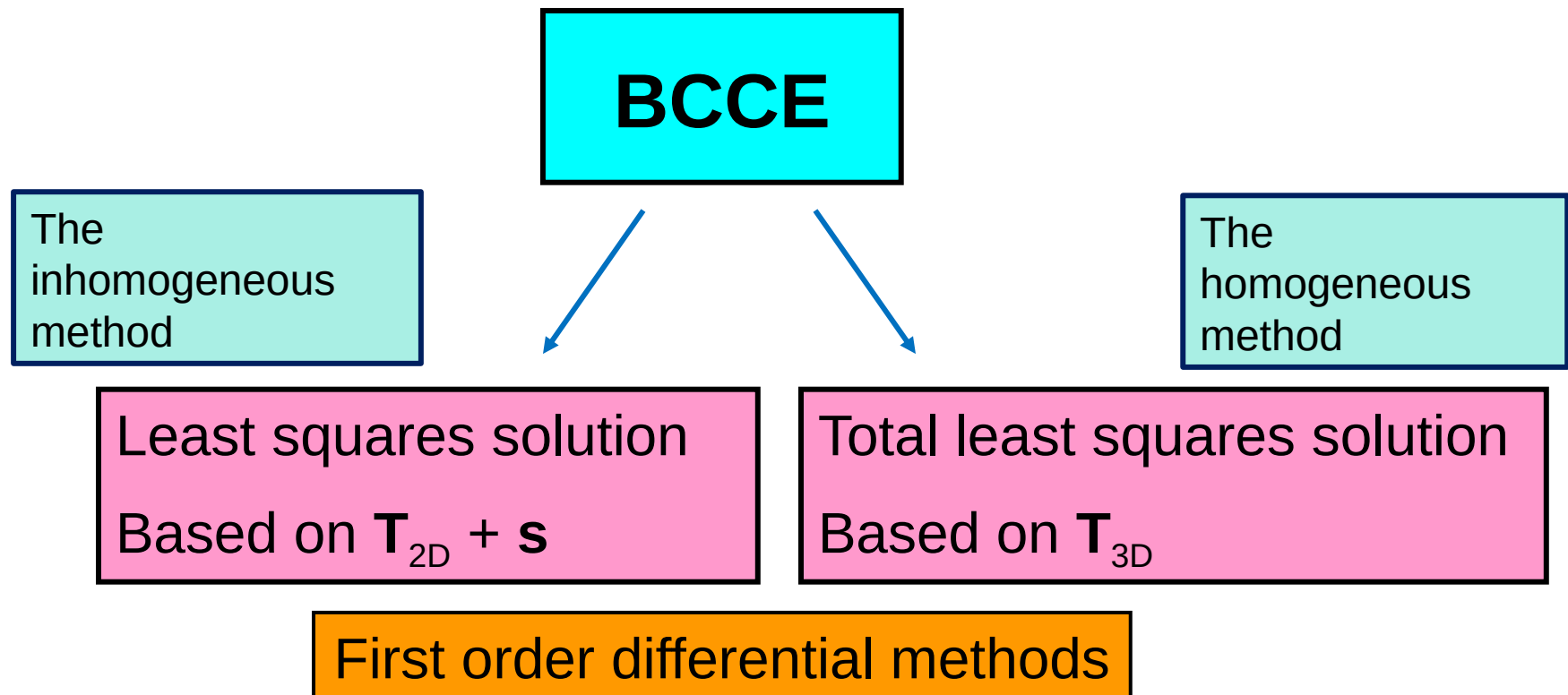


# TSBB15

# Computer Vision

Lecture 5  
Global motion estimation  
Tracking

# Motion estimation



# Motion estimation

- The techniques described next (and in the previous lecture) are suitable for determining an ***the optical flow***, an estimate of  $\mathbf{m}(\mathbf{x})$ , at each point  $\mathbf{x}$  in the image
- This is referred to as ***dense motion estimation***
  - Can still be characterized by a position dependent certainty measure
- An alternative is ***tracking***, where the motions of only a small set of points, or a single point, are determined
  - Later in this lecture...

# Motion estimation

- There are other approaches, for example

- Global smoothness of  $\mathbf{v}$  (Horn & Schunck)

Will be covered here

- Second order differential methods

- Parametric optical flow

- Et cetera

Will not be covered here

# The Horn & Schunck method

- At each point we seek the motion vector  $\mathbf{v} = (v_1, v_2)$  that satisfies the BCCE:

$$\frac{\partial I}{\partial t} + \frac{\partial I}{\partial u} v_1 + \frac{\partial I}{\partial v} v_2 = 0$$

- Problem: one equation but two unknowns
- Previously, we dealt with this problem by considering a *local* set of equations, assuming  $\mathbf{v}$  constant in a *local* region  $\Omega$
- Finding  $\mathbf{v}$  can also be dealt with by means of a *global* approach (with respect to the image)

# The Horn & Schunck method

- Let  $\mathbf{v}(u, v)$  denote the velocity vector field in an image, as a function of image position  $(u, v)$
- BCCE suggests that we should find  $\mathbf{v}(u, v)$  that minimizes

$$\epsilon = \int \left( \mathbf{v}(u, v) \cdot \nabla I + \frac{\partial I}{\partial t} \right)^2 d\mathbf{x}$$

Image gradient at  $(u, v)$

Time derivative at  $(u, v)$

Integration is now made over an entire image!

# The Horn & Schunck method

- We can (in principle) always find  $\mathbf{v}(u, v)$  that gives  $\varepsilon = 0$ :

$$\mathbf{v}(u, v) = -\frac{\partial I}{\partial t} \frac{\nabla I}{\|\nabla I\|^2} + \alpha(u, v) \begin{pmatrix} \frac{\partial I}{\partial v} \\ -\frac{\partial I}{\partial u} \end{pmatrix}$$

(why?)

Arbitrary function of  $(u, v)$

# The Horn & Schunck method

- Problem I:  
Singularities when  $\nabla I = \mathbf{0}$
- Problem II:  
Does not provide a unique solution since  $\alpha(u, v)$  can be arbitrary chosen
- Problem III:  
Strong variations in  $\nabla I$  may not correspond to strong variations in  $\mathbf{v}(u, v)$



# The Horn & Schunck method

- H&S 1981: Let's make  $\mathbf{v}(u, v)$  unique by adding a smoothness term to  $\varepsilon$
- This term should assure that  $\mathbf{v}(u, v)$  is as smooth as possible, seen as a function of  $(u, v)$
- Smoothness =  
“as little variation in  $\mathbf{v}$  as possible”

# The Horn & Schunck method

- H&S used a smoothness term:

$$\|\nabla v_1\|^2 + \|\nabla v_2\|^2$$

- Other types of smoothness terms appear in the literature

# The Horn & Schunck method

- New cost function

$$\begin{aligned} \epsilon = & \int \left( \mathbf{v}(u, v) \cdot \nabla I + \frac{\partial I}{\partial t} \right)^2 d\mathbf{x} \\ & + \lambda \int \left( \|\nabla v_1\|^2 + \|\nabla v_2\|^2 \right) d\mathbf{x} \end{aligned}$$

- The integrals are taken *over the entire image*
- $\lambda$  is a “smoothness weight”
- Our goal: find  $\mathbf{v}(u, v)$  that minimizes  $\epsilon$

# The Horn & Schunck method

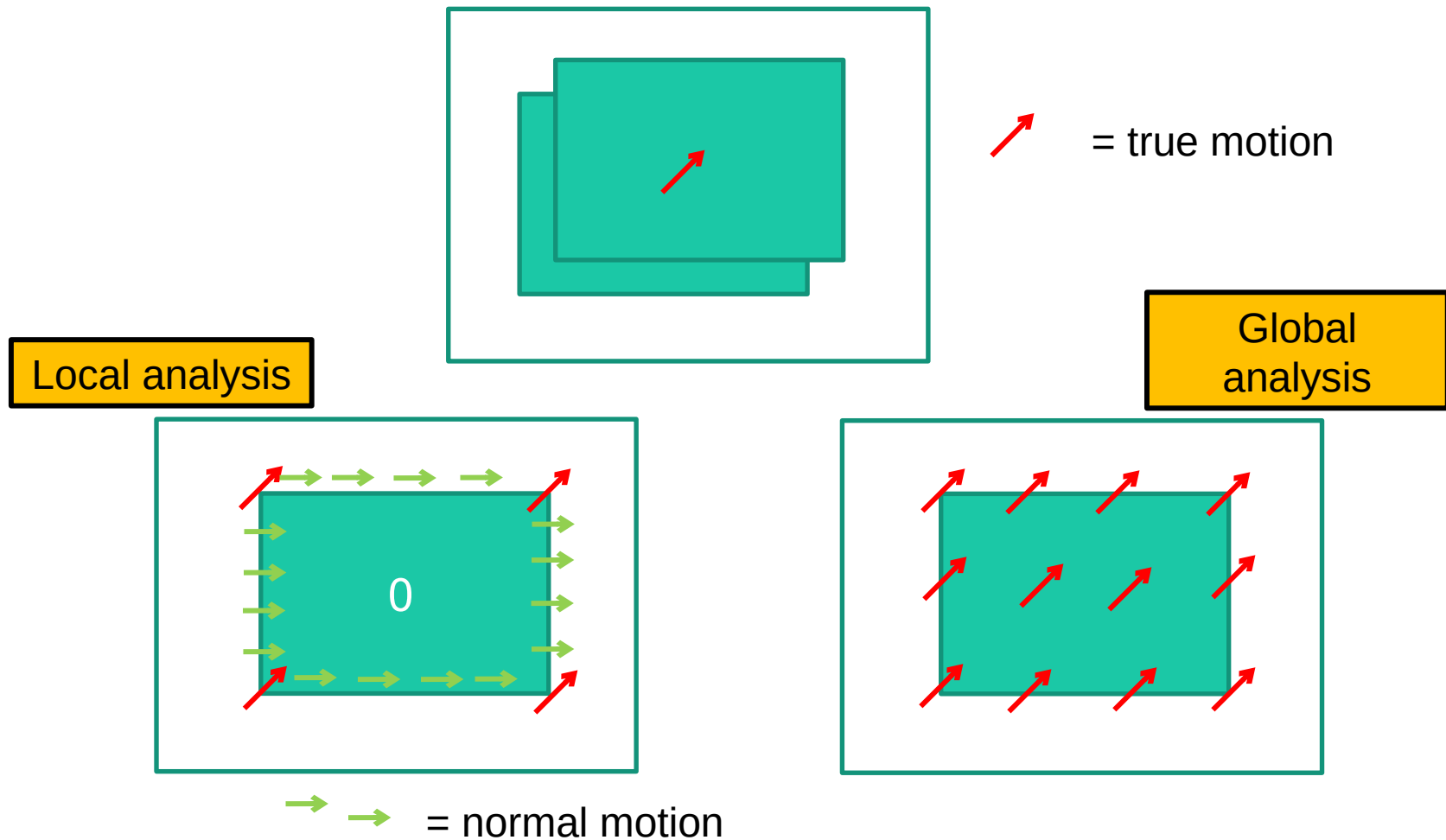
- This was one of the first established methods for motion estimation
- Often referred to as a “global” method
- Can (to some extent) deal with the aperture problem
- In practice:  $\mathbf{v}$  cannot be determined by solving a linear equation, instead iterative methods are required
  - Efficient algorithms exist
  - See e.g. D. Sun, et al, Secrets of Optical Flow Estimation and Their Principles, CVPR 2010.
- Not obvious how to choose  $\lambda$ 
  - constant or dependent on  $\mathbf{x}$ ?
- The smoothness constraint is not always valid
  - Sharp motion boundaries exist in practice
- More “sophisticated” methods use other types of smoothness terms

# The Horn & Schunck method

## NOTE!!

- Horn & Schunck's method is not correctly described in the book by R. Szeliski
  - In the printed book and e-book: on page 360, equation (8.70)
  - In the draft version on the web: on page 410, equation (8.70)
- The cost function  $E_{\text{HS}}$  lacks the regularization term

# Local vs global methods



## Second order differential methods

- Another approach for obtaining sufficient information to uniquely determine  $\mathbf{v}$  at each point is to differentiate BCCE again with respect to  $u$  and  $v$
- This method is again based on *local* computations

# Second order differential methods

- BCCE:

$$\frac{\partial I}{\partial t} + \frac{\partial I}{\partial u} v_1 + \frac{\partial I}{\partial v} v_2 = 0$$

- Differentiate with respect to  $u$  and  $v$ :

$$\begin{aligned} \frac{\partial^2 I}{\partial t \partial u} + \frac{\partial^2 I}{\partial u^2} v_1 + \frac{\partial^2 I}{\partial u \partial v} v_2 &= 0 \\ \frac{\partial^2 I}{\partial t \partial v} + \frac{\partial^2 I}{\partial u \partial v} v_1 + \frac{\partial^2 I}{\partial v^2} v_2 &= 0 \end{aligned}$$



## Second order differential methods

- Now we get 2 additional equations in variables  $\mathbf{v}(v_1, v_2)$ :

$$\mathbf{H}\mathbf{v} = -\frac{\partial}{\partial t}\nabla I$$

- $\mathbf{H}$  is the *Hessian matrix* (second order derivatives) of  $f$  w.r.t.  $u$  and  $v$
- Solve in a similar way as the LK-equation

# Multi order differential methods

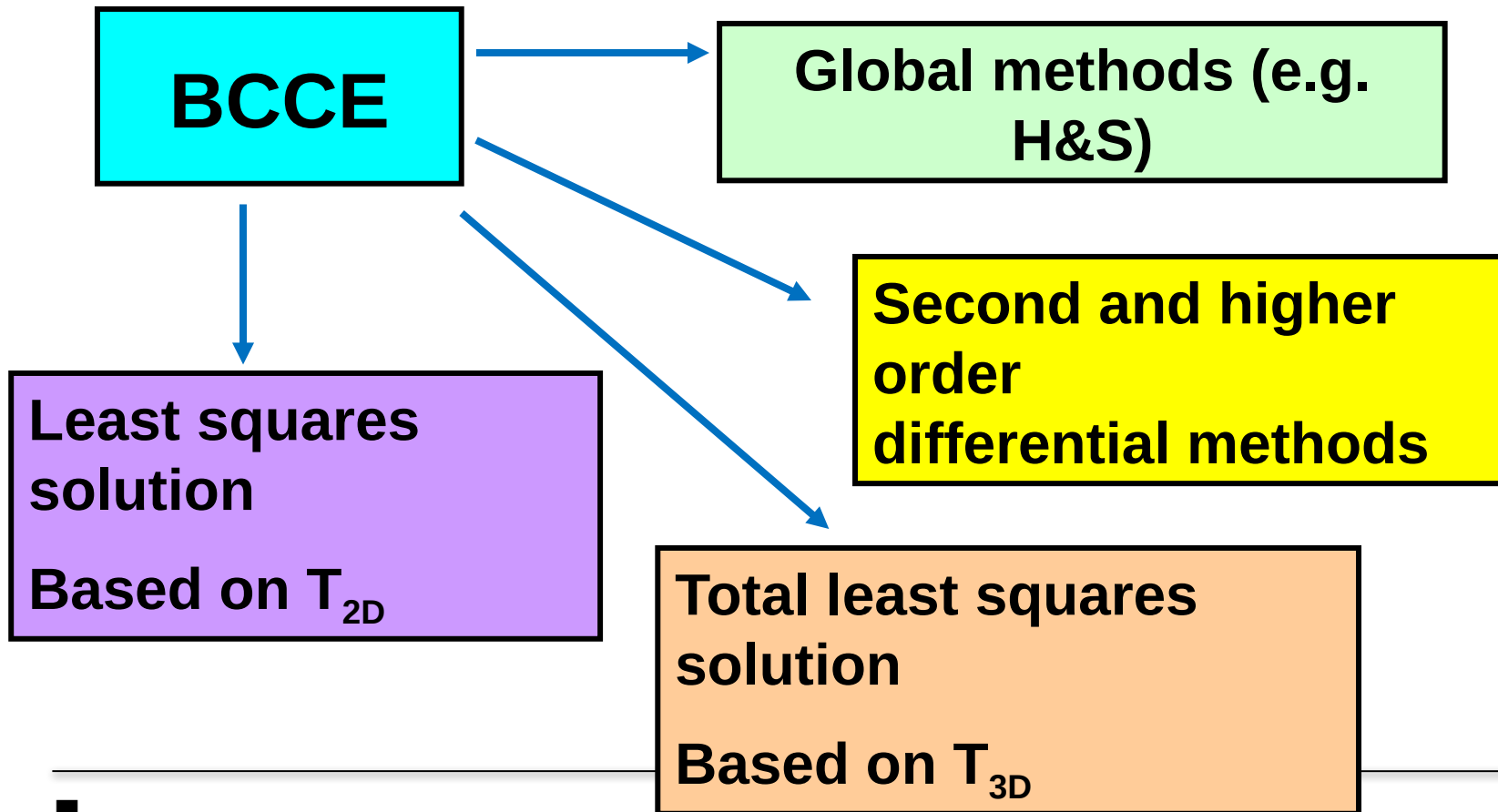
- There is nothing that prevents us from using both first and second order derivatives *simultaneously*!

$$\begin{pmatrix} \nabla^T I \\ \mathbf{H} \end{pmatrix} \mathbf{v} = - \begin{pmatrix} \frac{\partial I}{\partial t} \\ \frac{\partial}{\partial t} \nabla I \end{pmatrix}$$

# Multi order differential methods

- We get 3 (or more) equations and have 2 unknowns
- Solutions can still be found using various least squares techniques  
(how?)

# Motion estimation, summary



# Motion estimation, summary

- In the ideal case, all methods (in principle) should give the same solution
- They differ mainly with respect to
  - Sensitivity to
    - noise
    - deviations from model assumptions
  - Computational demand
  - Certainty measures
- For all methods: different sizes of  $\Omega$  and different ways to estimate gradients give different quality of results

# Advanced variations of basic methods

- These basic methods for motion estimation, in particular the local ones, can be significantly improved (at moderate cost) by using one or more ***advanced techniques***, such as
  - Refinement iterations
  - Course-to-fine refinement
  - Spatial filtering of motion estimates
  - Robust error norms
  - Symmetry in  $I$  and  $J$
  - Affine transformation

# Refinement iterations

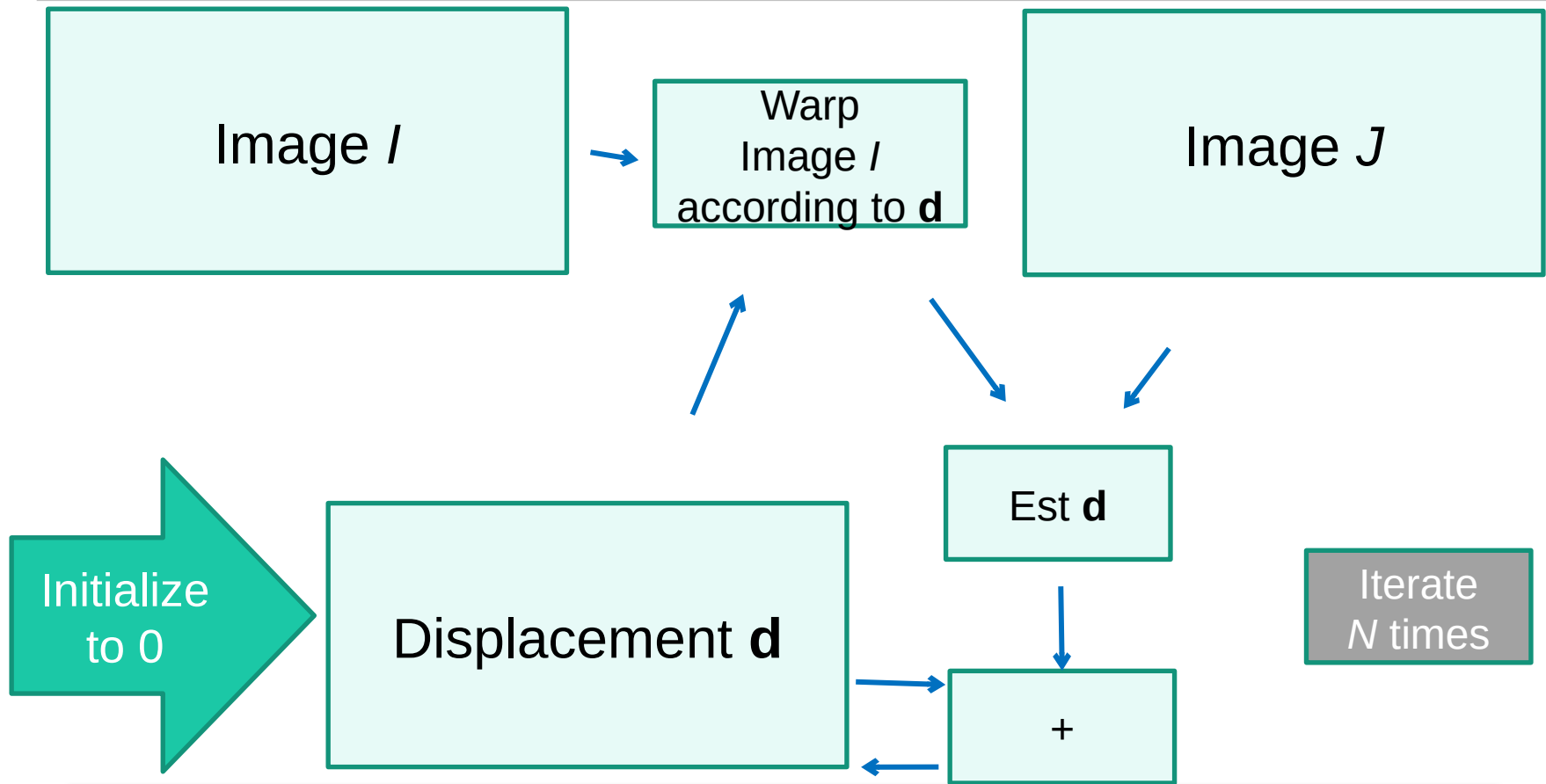
- The basic methods described here are based on a set of assumptions, e.g.:
  - Brightness constancy: e.g., for 2-image case:
$$J(\mathbf{x}) = I(\mathbf{x} + \mathbf{d})$$
  - High order terms in Taylor expansions can be neglected
  - Constant  $\mathbf{d}$  (or  $\mathbf{v}$ ) within  $\Omega$
- In general these assumptions are not all correct: estimate of  $\mathbf{d}$  (or  $\mathbf{v}$ ) is inaccurate

# Refinement iterations

- The estimate  $\mathbf{d}$  (or  $\mathbf{v}$ ) should, however, in most cases be approximately correct
- Warp  $I$  in accordance to estimated  $\mathbf{d}$  (or  $\mathbf{v}$ )
  - If  $\mathbf{v}$  is correctly estimated, the two images are more or less equal
  - If not, there is some remaining  $\mathbf{d}$  (or  $\mathbf{v}$ ) that can be estimated from the new  $I$  and the old  $J$
  - Iterate  $N$  times and accumulate new estimates of  $\mathbf{v}$  (refine  $\mathbf{v}$ ) in each iteration



# Refinement iterations



# Refinement iterations

- $N$  = number of iterations, depends on the application and on the data (images)
- Does not have to be very large
- For most applications: a “few” iterations are often sufficient

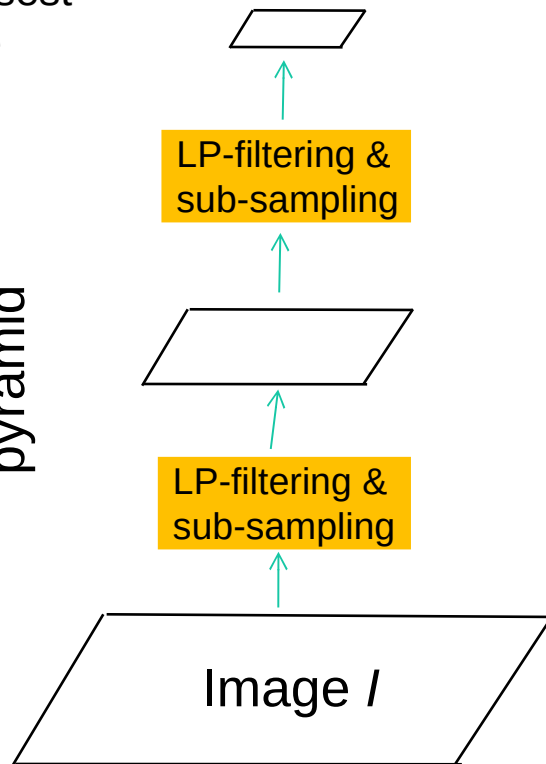
# Coarse-to-fine refinement

- In local motion analysis, the motion of each point is analyzed within a region  $\Omega$ 
  - $\Omega$  has some radius  $R$
- $\mathbf{d}$  cannot be robustly determined if  $|\mathbf{d}| > R$
- $R$  cannot be made too large:
  - $\mathbf{d}$  will not be constant in  $\Omega$
  - Taylor expansion of  $I(\mathbf{x} + \mathbf{y} + \mathbf{d})$  not only linear
- To deal with larger  $\mathbf{d}$ , use coarse-to-fine refinement based on scale pyramids
  - See lecture 2

# Coarse-to-fine refinement

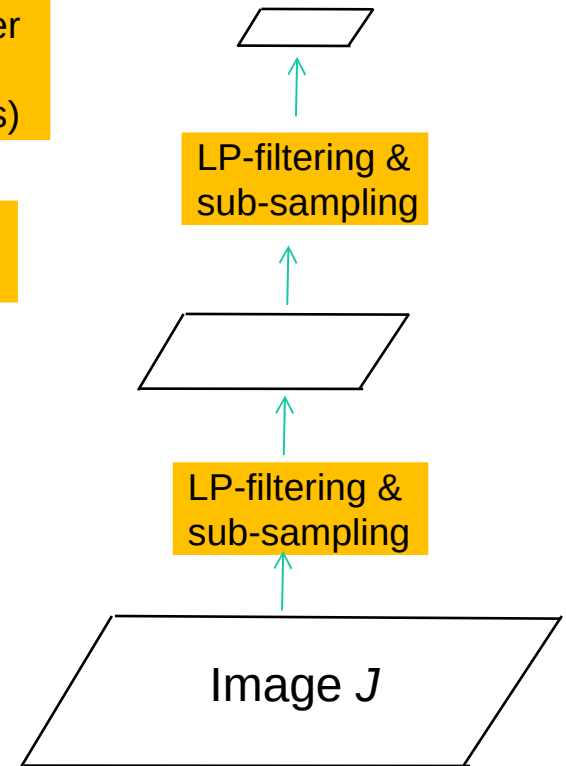
Coarsest  
scale

Gaussian  
pyramid



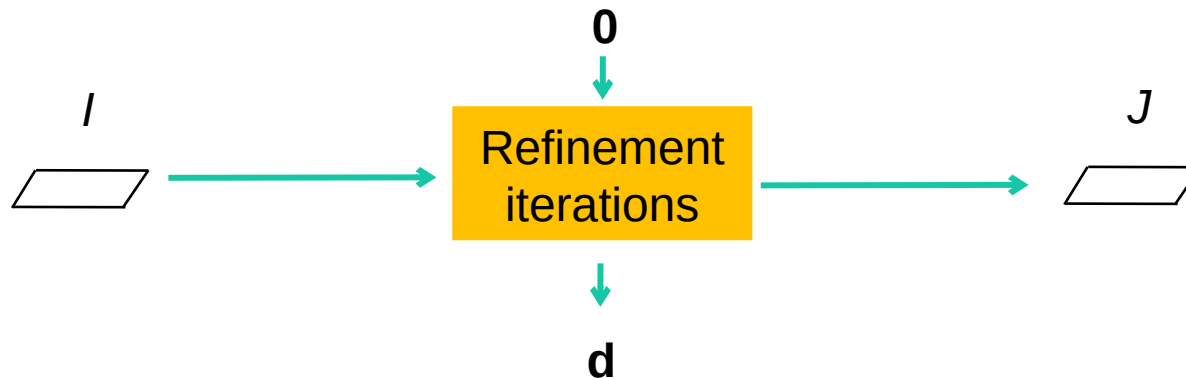
Down-sample by a factor  
2 in both directions. Other  
factors can also be used  
(even non-integer factors)

Number of scale levels  
is application dependent



# Coarse-to-fine refinement

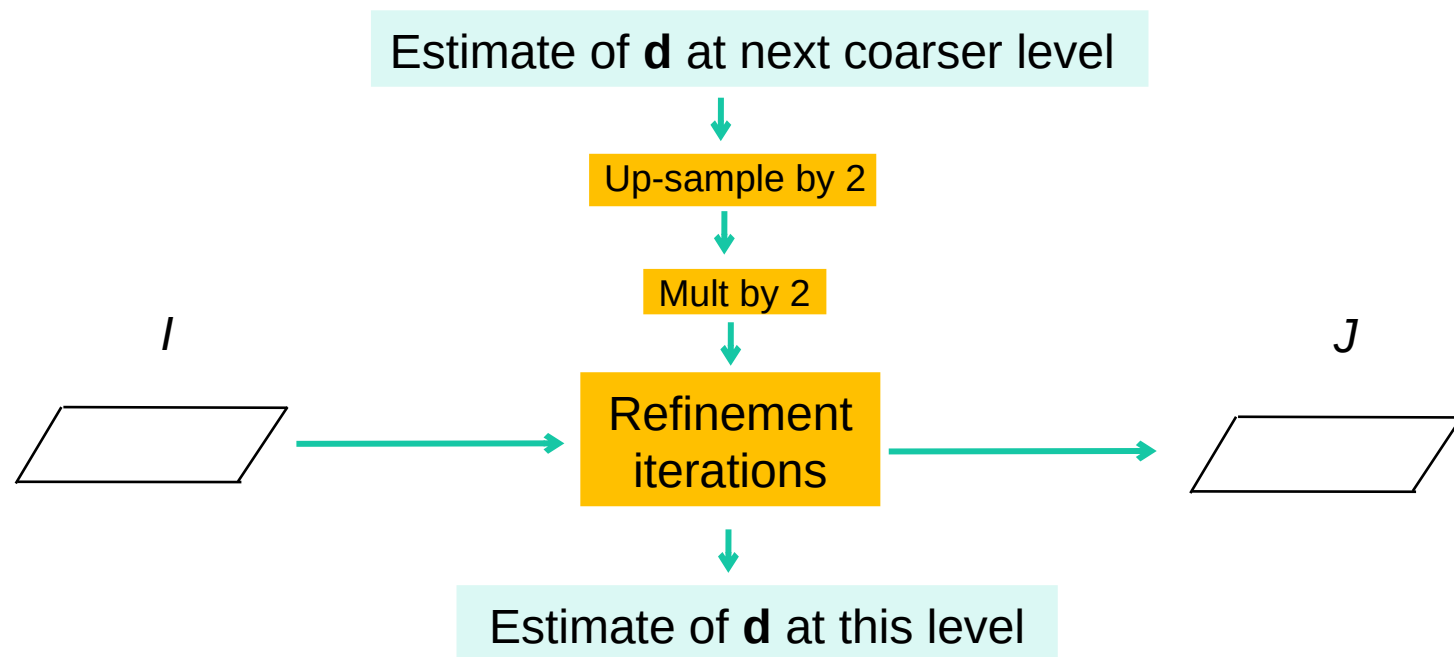
- Start at the coarsest level
- Perform refinement iterations where  $\mathbf{d}$  is initiated to  $\mathbf{0}$  at all points
- Produces an initial estimate of  $\mathbf{d}$  at this level



## Coarse-to-fine refinement

- This initial estimate of  $\mathbf{d}$  is then up-sampled to fit the image size at the next finer level
- Also:  $\mathbf{d}$  is multiplied by 2 (or suitable factor) since displacements at the next finer level are 2 times as large as at the previous level
- Use this new  $\mathbf{d}$  as initial estimate in refinement iterations at the finer level

# Coarse-to-fine refinement



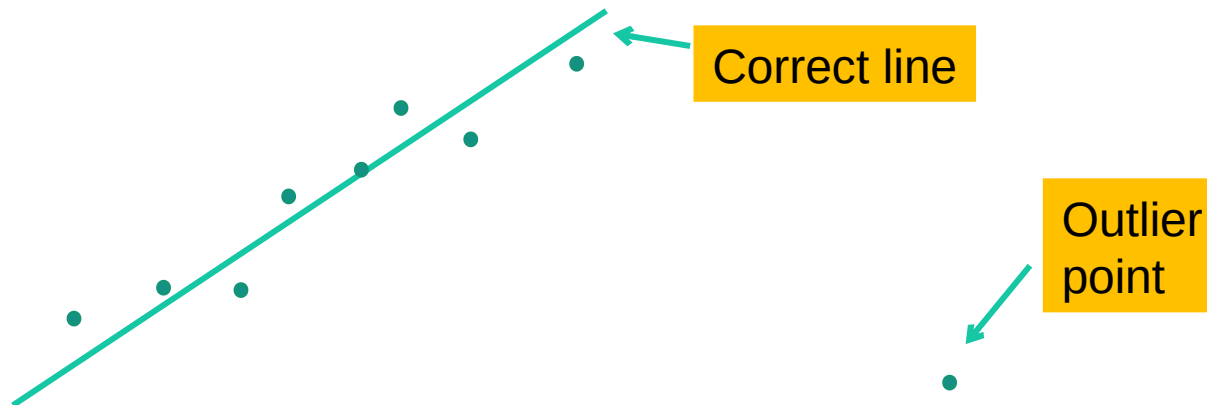
# Coarse-to-fine refinement

- Continue this processing from the coarsest level all the way to the finest level
- Estimate of  $\mathbf{d}$  from the finest level is the final estimate from this coarse-to-fine processing
- Can manage magnitudes of  $\mathbf{d}$  which are in the order of  $R$  for  $\Omega$  at the coarsest level
- Note: estimates of  $\mathbf{d}$  at a coarser level does not have to be ***very accurate***, it will be refined at the next finer level!



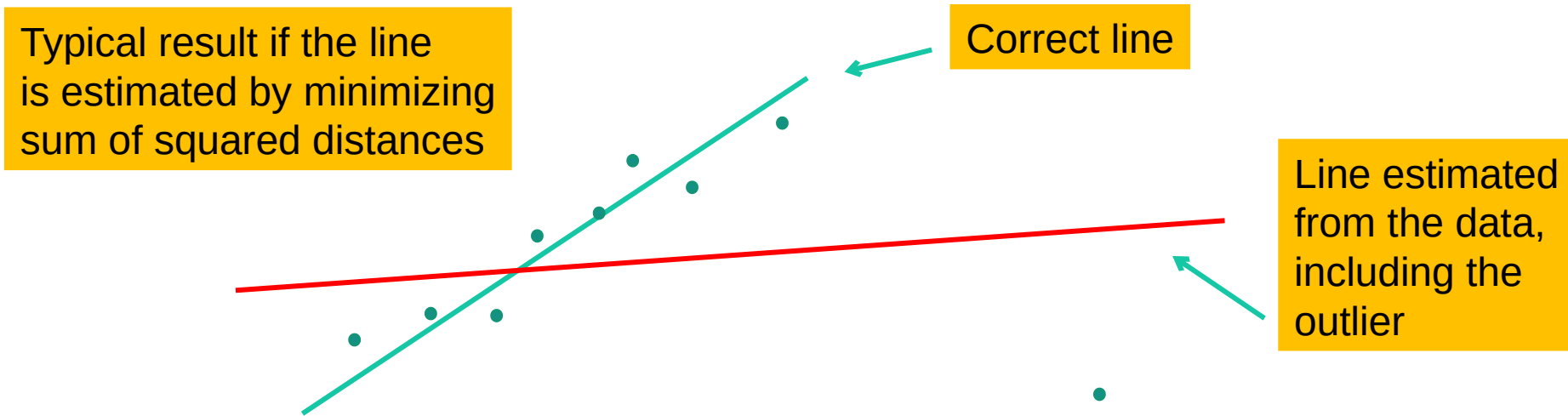
# Outliers

- Definition: an **outlier** is a point (or data entry) that doesn't fit the model assumed for the data
  - Data that fits the model: **inliers**
- Example: fitting a line to a set of points



# Outliers

- If outliers are allowed to affect estimation of a model in the same way as inliers, the model can become very distorted



# Spatial filtering of motion estimates

- Motion estimates at two adjacent pixels should often be very similar
  - The points are projections of 3D points on the same rigid object
  - Not true at ***motion boundaries!***
- Motion estimates can also be degraded by
  - Image noise
  - Invalid assumptions (e.g., because of outliers)

# Spatial filtering of motion estimates

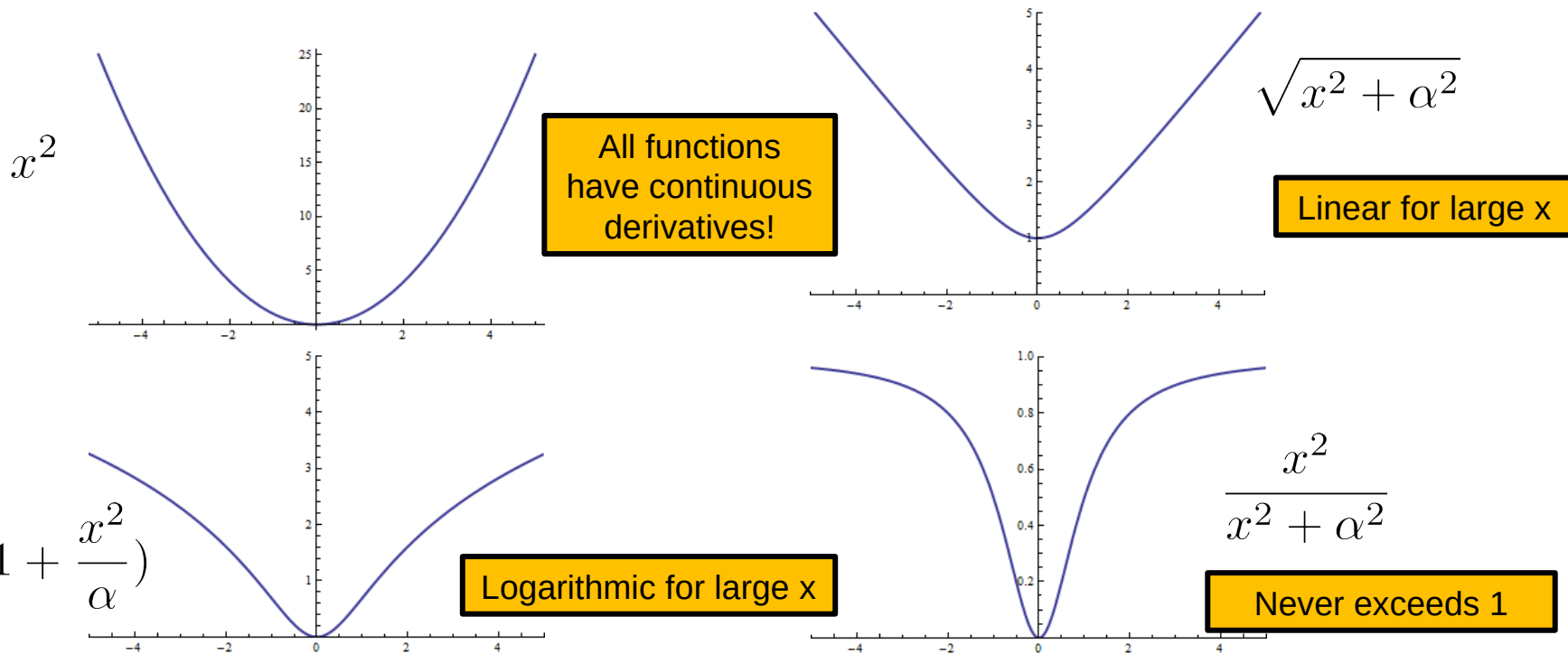
- To reduce these effects it makes sense to allow the estimate of  $\mathbf{d}$  to be affected by its neighbors
  - Local averaging, weighted by a spatial window
  - Corresponds to LP-filtering of  $\mathbf{d}$
- Even better: use normalized convolution
  - Takes certainty of  $\mathbf{d}$  into account
- Alternatively: use median filtering
  - Avoids large influence from ***outliers***

# Robust errors

- Adding squared distances implies:  
***Computing a weighted average of the distances, where each weight = the distance***
- Implies: outliers are given a high weight
  - Not what we want!!
- This effect can be reduced by using ***robust errors***

# Robust errors

- Replace the square function with alternative function, for example



# Symmetric formulation

- The 2-image version of the LK-method does not treat images  $I$  and  $J$  in the same way
  - Spatial gradients are only computed in  $I$
  - In refinement iterations, only one image is warped
- In the ideal situation, swapping  $I$  and  $J$  should produce a consistent result
  - Not always true

# Symmetric formulation

- Use a symmetric formulation:

$$J(\mathbf{x} - \mathbf{d}/2) = I(\mathbf{x} + \mathbf{d}/2)$$

instead of

$$J(\mathbf{x}) = I(\mathbf{x} + \mathbf{d})$$



# Symmetric formulation

- Finding  $\mathbf{d}$  as the minimizer of

$$\epsilon = \int_{\Omega_0} w(\mathbf{y}) (I(\mathbf{x} + \mathbf{y} + \mathbf{d}/2) - J(\mathbf{x} + \mathbf{y} - \mathbf{d}/2))^2 d\mathbf{y}$$

- Can be solved in a similar way as before:

$$\mathbf{T} \mathbf{d} = \mathbf{s}$$

$\mathbf{T}$  and  $\mathbf{s}$  contain  
gradients from  
both  $I$  and  $J$


(how?)

# Affine transformation

- The local motion model for the 2 image case only includes a translation:

$$J(\mathbf{x}) = I(\mathbf{x} + \mathbf{d})$$

- A more complex model could also include an affine transformation:


$$J(\mathbf{x}) = I(\mathbf{A} \mathbf{x} + \mathbf{d})$$

Unknown parameters  
to be estimated,  
depend on  $\mathbf{x}$

# Affine transformation

- $\mathbf{A}$  is a  $2 \times 2$  matrix
- In practice, set  $\mathbf{A} = \mathbf{I} + \mathbf{A}'$ 
  - $\mathbf{A}'$  is then often a small matrix, easier to estimate

- Set
 
$$\mathbf{A}' = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \\ d_1 \\ d_2 \end{pmatrix}$$

and minimize  $\varepsilon$  over  $\mathbf{z}$  (how?)

- Leads to  $\mathbf{T}' \mathbf{z} = \mathbf{s}'$

$\mathbf{T}'$  is  $6 \times 6$   
 $\mathbf{s}'$  is 6-dimensional

# Tracking

Image at  $t = t_0$

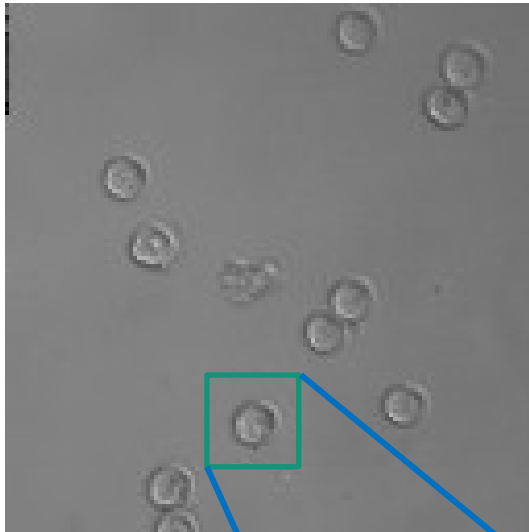


Image at  $t = t_0 + \Delta t$

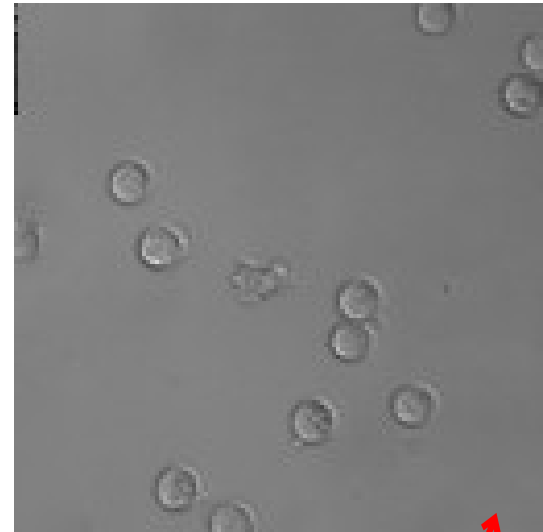
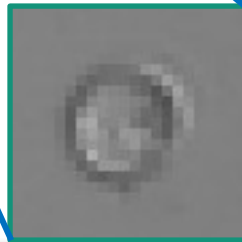


Image *template*  
that we want to find  
in the *target* image



# Tracking vs. motion estimation

- In ***motion estimation***, the motion field  $\mathbf{m}(\mathbf{x})$  is estimated either as a displacement field  $\mathbf{d}(\mathbf{x})$  between two images, or as a velocity field  $\mathbf{v}(\mathbf{x})$  based on a continuous time model
  - The result is  $\mathbf{d}(\mathbf{x})$  (or  $\mathbf{v}(\mathbf{x})$ ) as a function of  $\mathbf{x}$  for all image points
- In ***tracking***, we determine  $\mathbf{d}(\mathbf{x})$  (or  $\mathbf{v}(\mathbf{x})$ ) for a single point, or for a region  $\Omega$  around this point (the template)
  - The result is  $\mathbf{d}$  (or  $\mathbf{v}$ ) for this template



## Tracking vs. estimation of $m(x)$

- Tracking can also be applied to a smaller set of points (templates) determined as interesting to track
  - As a consequence, tracking can be done with low computational cost, alternative it allows more complex methods to be used since they are not applied to every image point
- Typically, tracking of a template is made over several consecutive images in an video sequence
  - As long as the template can be robustly re-identified in each target image

# Applications for tracking

Tracking can be used for

- Following specific objects in an image sequence
  - People, vehicles, targets, etc
- For efficiency:
  - assume small  $\mathbf{v}$  between each image
- Producing ***point correspondences*** for specific interest points in two or more images of the same scene
  - *Structure from motion*
  - *Ego-motion estimation*
- Determine 3D motion based on motion in the image
- Segmentation based on distinct objects moving with distinct motions
- Stereo matching (original app for LK-tracking!)
- Video compression



# Basic tracking methods

- See tracking as a special case of 2-image motion estimation where image  $I$  is the template, and image  $J$  is an image from a video sequence (the target image) (or the other way around)
  - Use the LK-approach, or other local methods for motion estimation.
  - Referred to as ***LK-tracking***
  - Use the advanced methods mentioned previously
    - In particular refinement iterations and scale pyramids
  - Can be efficiently implemented in software & hardware
    - GPGPU (Graphics hardware)

# Basic tracking methods

- See tracking as the problem of re-identifying a template in a target image
  - Block matching (grid-based method)
- See tracking as the problem of re-identifying a “blob” of pixels that have been determined as “not background”
  - See subsequent lectures

# Block matching

A rather straight-forward approach:

- Given
  - A template  $\Omega$
  - A target image  $J$
  - A predicted position of  $\Omega$  in  $J$
  - A range  $N$
- Prediction can be: where  $\Omega$  was found in the previous image in the sequence
  - Can also include statistical models (Kalman filter)
- Extract a set of regions in  $J$  around  $\mathbf{x}$ , of same size as  $\Omega$ 
  - For example, in the ranges  $(x_1 \pm N/2, x_2 \pm N/2)$
  - Typically with integer shifted displacements
  - Number of patches is in the order of  $N^2$

# Block matching

- Compare the template with all patches, find best match
  - We need some similarity measure to do this!
  - Generates a matching function  $\varepsilon(d_1, d_2)$
  - Find minimum of  $\varepsilon$ , (or maximum, depending on how  $\varepsilon$  is defined)
  - Its position in  $J$  is  $(x_1 + d_1, x_2 + d_2)$ ,  $-N/2 \cdot d_1, d_2 \cdot N/2$
  - The estimated displacement of the template between image 1 and image 2 given by  $(d_1, d_2)$
- Referred to as *block matching* or *template matching*
- Can be implemented efficiently on **GPGPU hardware**

# Block matching

Some issues that need to be resolved

- How do we compare patches (=blocks of pixels)? Examples:
  - Sum of squared differences (SSD)
  - Sum of absolute differences (SAD)
  - Cross-correlation (CC), normalized cross-correlation (NCC)
- How do we choose a reasonable  $N$ ?
  - Must be large enough to cover the displacements that occur for the application
  - Computational complexity grows with  $N^2$
- Best match may not be for a unique displacement
  - Repetitive patterns
- Sub-pixel accuracy
  - $\varepsilon(d_1, d_2)$  can be interpolated to determine inter-pixel optima

# Good features to track

- A paper by Tomasi & Kanade analyzes *which* templates are feasible for tracking
- Conclusion: we should consider templates that give  $\mathbf{T}_{2D}$  which are definitely non-singular (**big surprise?**)
- T&K propose that  $\min(\lambda_1, \lambda_2) > \text{threshold}$  is a useful criteria for template selection

# Tomasi-Kanade

- The TK-criteria can be used to find *interest points* in an image, i.e., points that easily can be identified in several images
- In some applications we may be interested in tracking all such interest points
- Compare to the Harris-detector

# Practical aspects of tracking

## Template update

- 3D objects tend to change appearance over time when moving in a scene
  - Change of aspect and apparent size relative to the camera
- Suggests that the template should be updated from the target image, e.g.,
  - At regular time intervals
  - When the matching measure degrades too much
- Tricky to implement robustly
  - Difficult to avoid that  $\Omega$  starts to contain the background instead of the relevant object



# Practical aspects of tracking

## Track-retrack

- 3D Tracking of an object over  $N$  images creates a motion trajectory, from image 1 to image  $N$ 
  - A “curve” defined by the image coordinates  $\mathbf{x}(k)$  of where  $\Omega$  is found in each image,  $k = 1, \dots, N$
- Generated by starting at  $\mathbf{x}(1)$  in image 1 and successively finding the position of  $\Omega$  in each new,  $\mathbf{x}(k)$ , image ***forward in time***
- Ideally, if we instead start in image  $N$ , at position  $\mathbf{x}(N)$ , and track  $\Omega$  ***backward in time***, we should end up at  $\mathbf{x}(1)$
- If the forward and backward trajectories differ too much, the tracking can be considered as failed, cannot be trusted for further processing

# Practical aspects of tracking

## In the literature

- The basic LK-based methods (gradient based) appear in the literature under a variation of names, e.g.,
  - Lucas-Kanade (LK)
  - Kanade-Lucas (KL)
  - Lucas-Kanade-Tomasi (LKT), or permutations
  - Shi-Tomasi (ST)
- Can also be used as a refinement after block matching