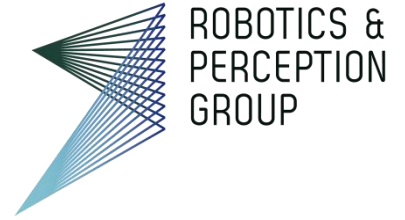




University of  
Zurich <sup>UZH</sup>

**ETH** zürich

Institute of Informatics – Institute of Neuroinformatics



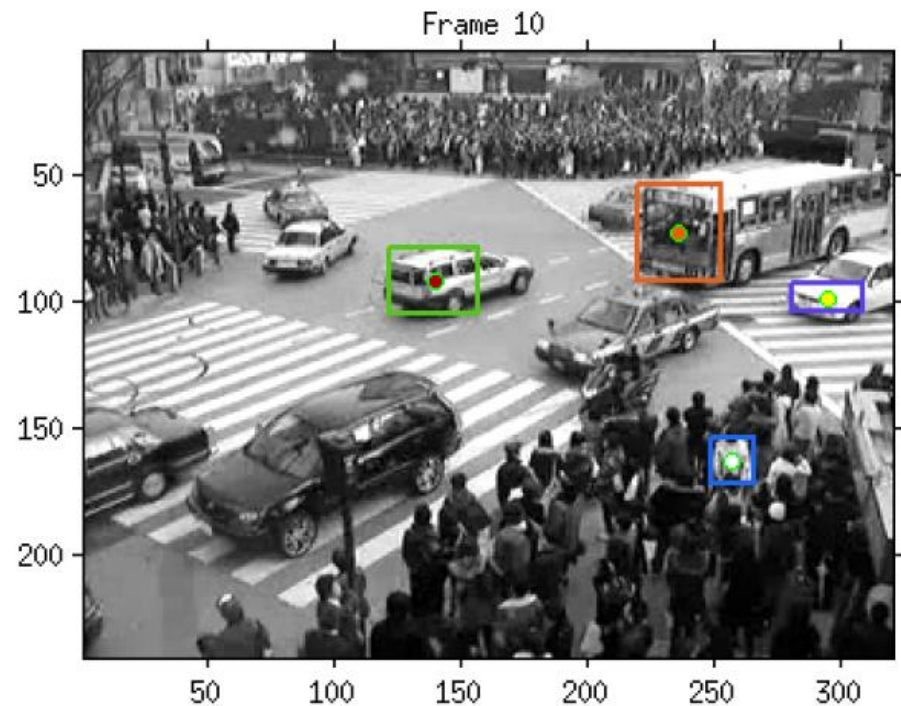
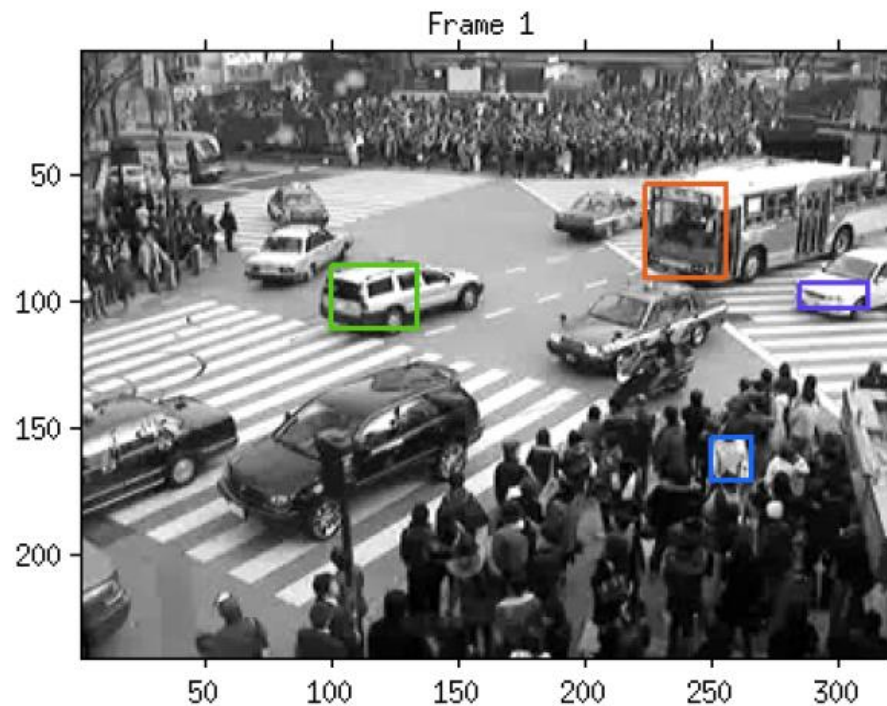
# Lecture 11

## Tracking

Davide Scaramuzza

# Lab Exercise 7 - Today

- Room ETH HG E 1.1 from 13:15 to 15:00
- Work description: Lucas-Kanade template tracking

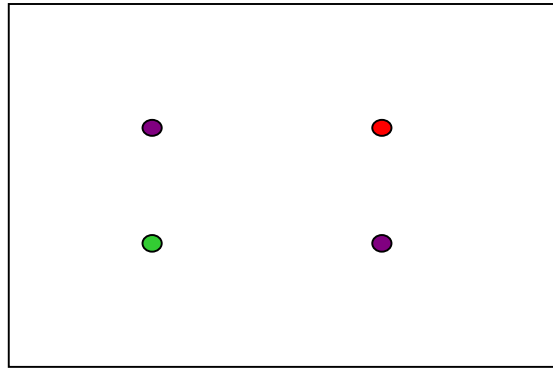


# Outline

- Point tracking
- Template tracking
- Tracking by detection of local image features

# Point Tracking

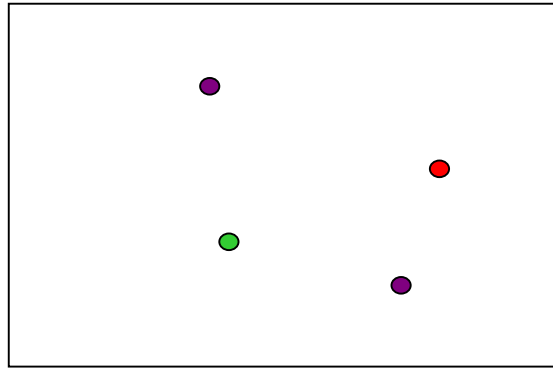
- **Problem:** given two images, estimate the motion of a pixel point from image  $I_0$  to image  $I_1$



$I_0(x, y)$

# Point Tracking

- **Problem:** given two images, estimate the motion of a pixel point from image  $I_0$  to image  $I_1$

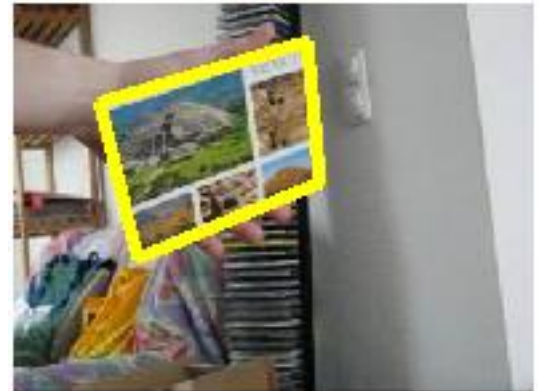
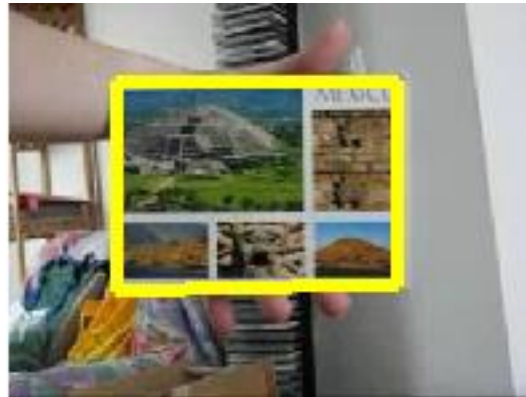


$I_1(x, y)$

# Template tracking

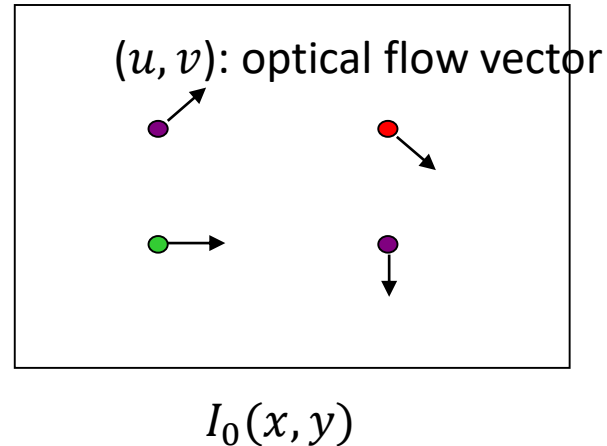
- **Problem:** given two images, estimate the *warping* that defines the *motion* and/or *the distortion* of a *template* from image  $I_0$  to image  $I_1$

Template image



# Point Tracking

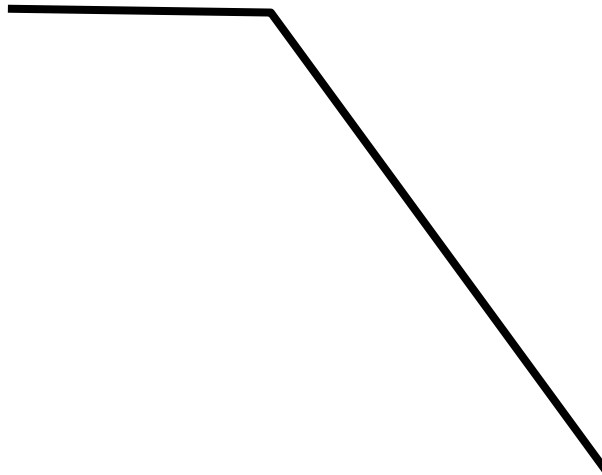
- **Problem:** given two images, estimate the motion of a pixel point from image  $I_0$  to image  $I_1$



- Two approaches exist, depending on the amount of motion between the frames
  - **Block-based methods**
  - **Differential methods**

# Point Tracking

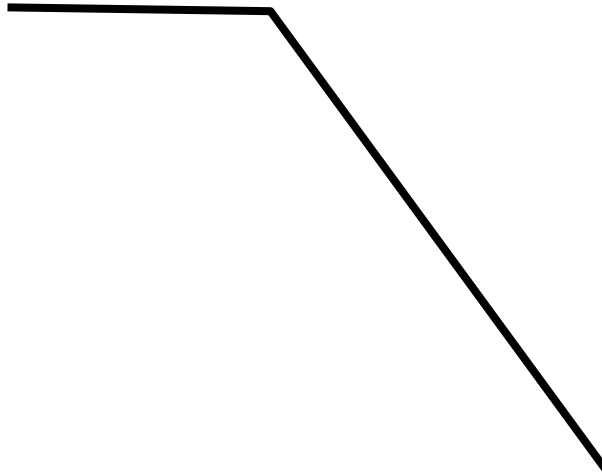
- Consider the motion of the following corner





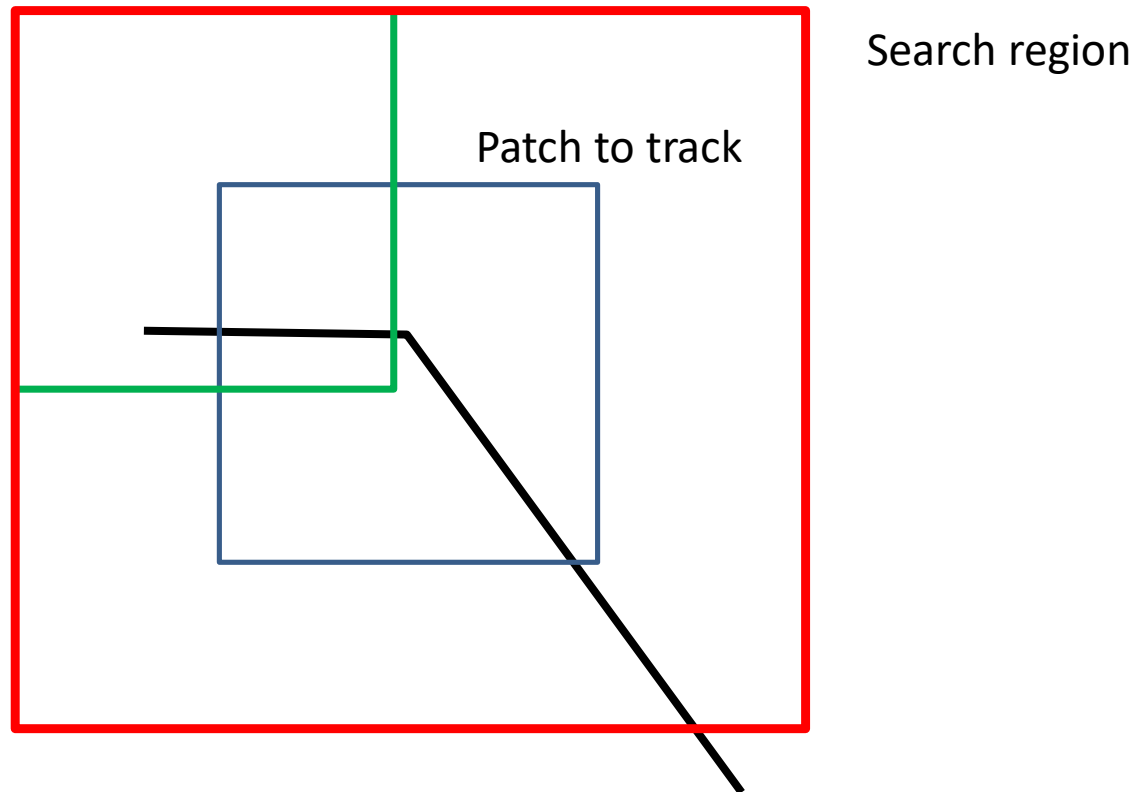
# Point Tracking

- Consider the motion of the following corner



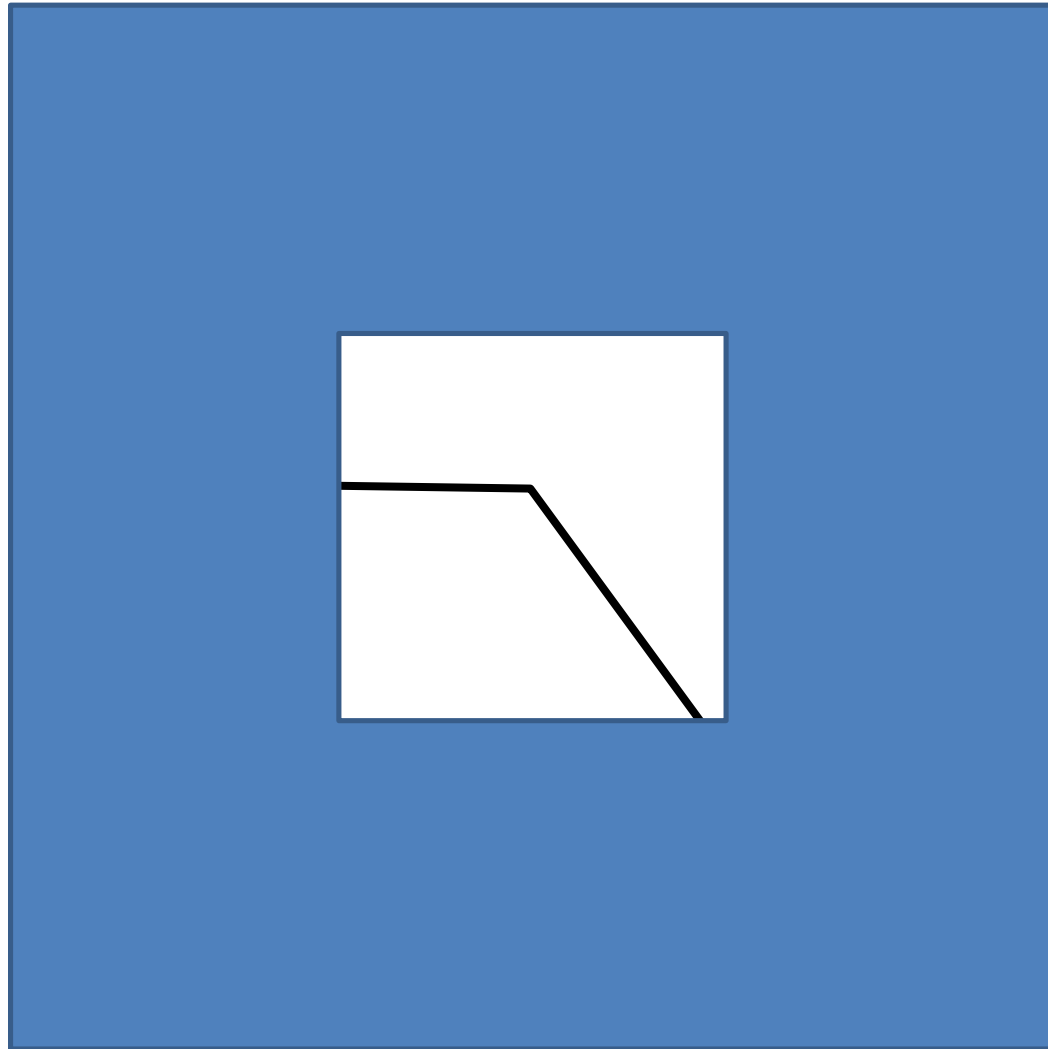
# Point Tracking with Block Matching

- Search for the corresponding patch in a **neighborhood around** the point.
- Use SSD, SAD, or NCC to search for corresponding patches in a local neighborhood of the point. The search region is usually a  $D \times D$  squared patch.



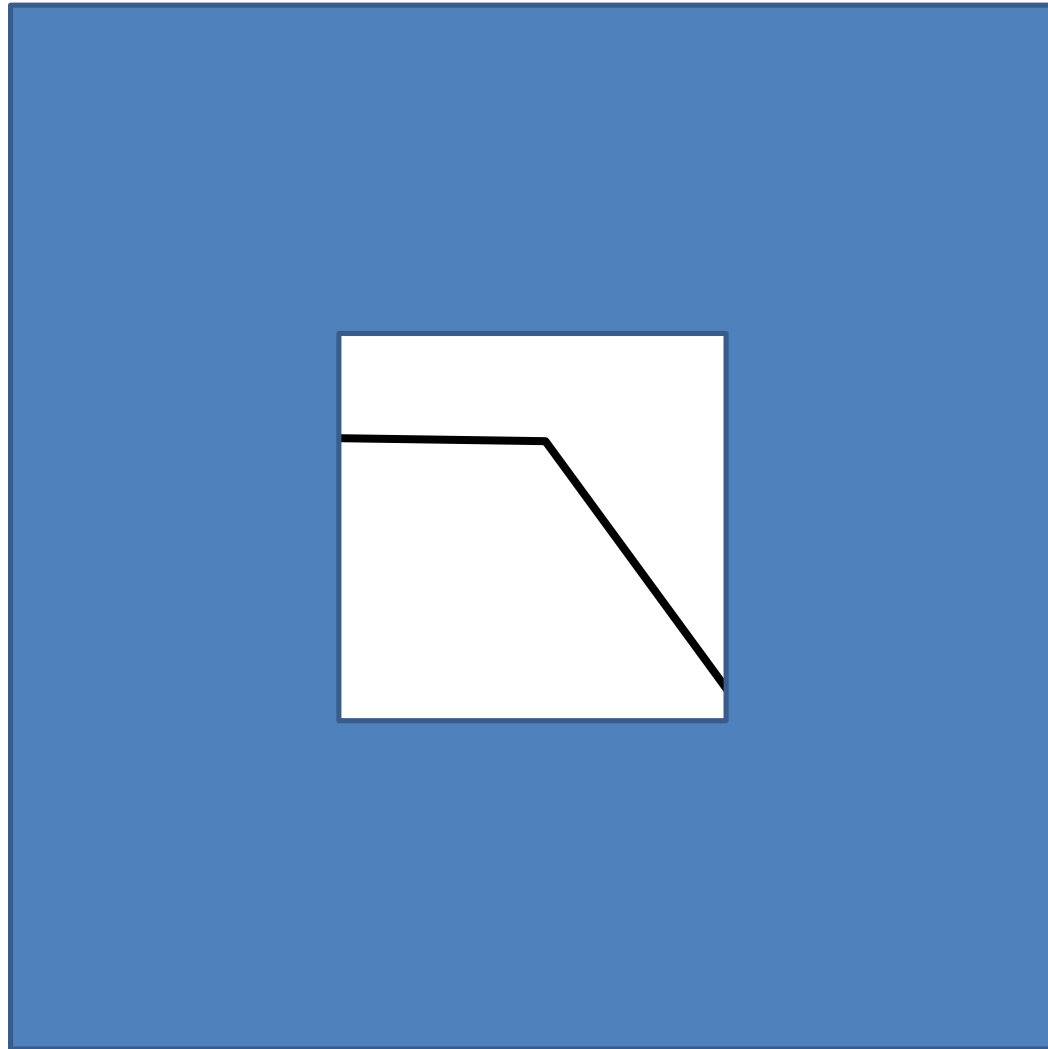
# Point Tracking with Differential Methods

- Looks at the local brightness changes at the **same** location. **No patch shift** is performed!



# Point Tracking with Differential Methods

- Looks at the local brightness changes at the **same** location. **No patch shift** is performed!

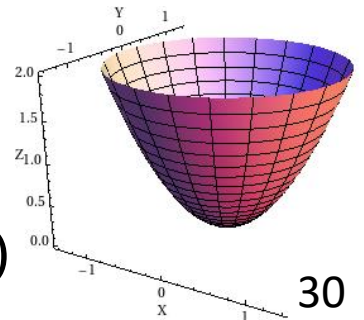


# Applying the Spatial Coherency

- Assume that all the pixels in the patch undergo the same motion (same  $u$  and  $v$ ) (usually, a square patch of  $n \times n$  pixels is used)
- Also, assume that the time interval between the two images  $I_0, I_1$  is small
- We want to find the motion vector  $(u, v)$  that minimizes the Sum of Squared Differences (SSD):

$$\begin{aligned} SSD &= \sum (I_0(x, y) - I_1(x + u, y + v))^2 \\ &\cong \sum (I_0(x, y) - I_1(x, y) - I_x u - I_y v)^2 \\ &= \sum (\Delta I - I_x u - I_y v)^2 \end{aligned}$$

This is a simple quadratic function in two variables  $(u, v)$



# Computing the Motion Vector

$$E = \text{SSD} = \sum (\Delta I - I_x u - I_y v)^2$$

- To minimize the E, we differentiate E with respect to  $(u, v)$  and equate it to zero

$$\frac{\partial E}{\partial u} = 0, \quad \frac{\partial E}{\partial v} = 0$$

$$\frac{\partial E}{\partial u} = 0 \Rightarrow -2 \sum I_x (\Delta I - I_x u - I_y v) = 0$$

$$\frac{\partial E}{\partial v} = 0 \Rightarrow -2 \sum I_y (\Delta I - I_x u - I_y v) = 0$$

# Computing the Motion Vector


$$\frac{\partial E}{\partial u} = 0 \Rightarrow -2 \sum I_x(\Delta I - I_x u - I_y v) = 0$$

$$\frac{\partial E}{\partial v} = 0 \Rightarrow -2 \sum I_y(\Delta I - I_x u - I_y v) = 0$$

- Linear system of two equations in two unknowns
- We can write them in matrix form:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}$$

Notice that these are  
NOT matrix products  
but **pixel-wise**  
products!


$$\Rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}^{-1} \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}$$

**M matrix**

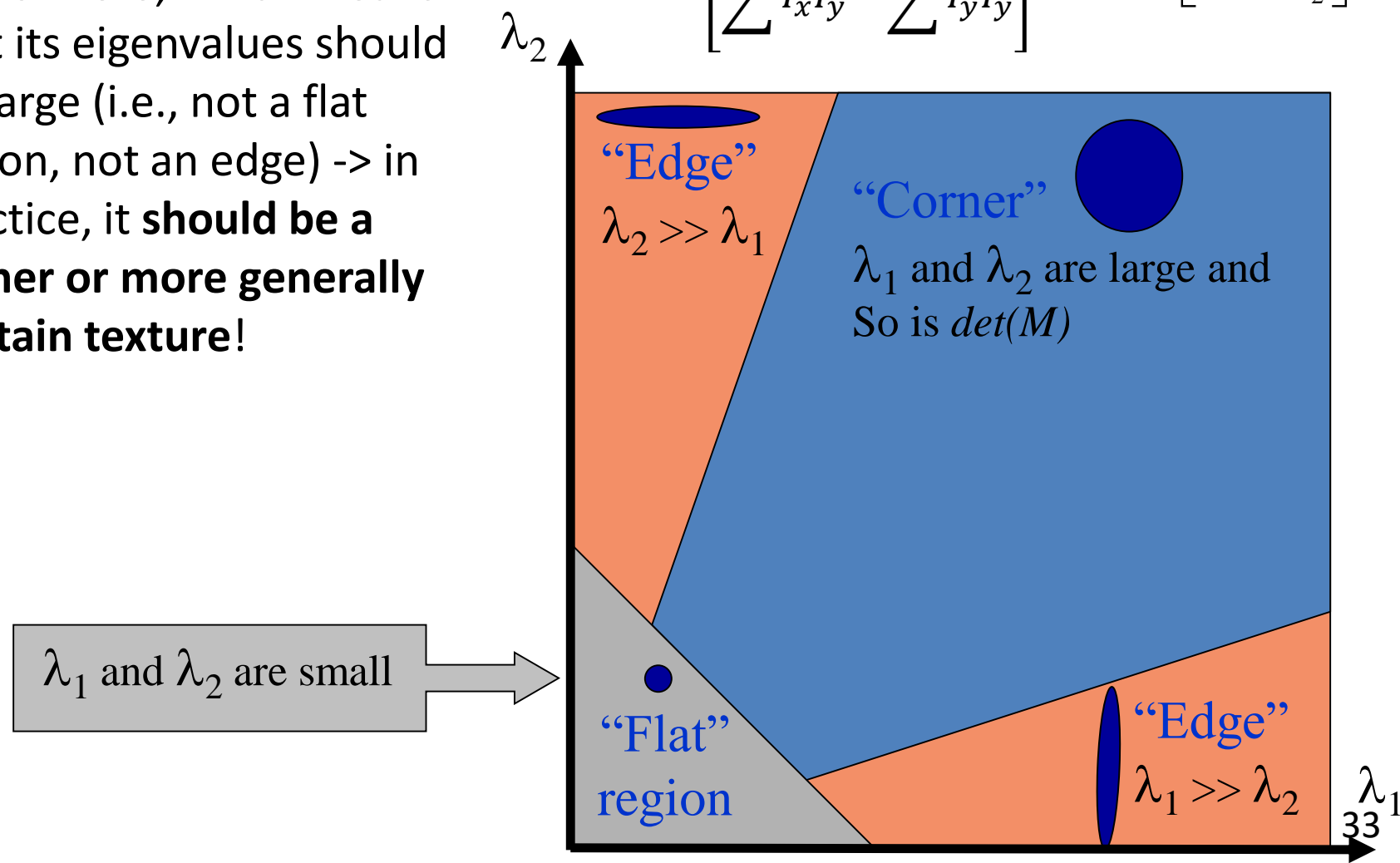
**Haven't we seen this matrix already?**

**Recall Harris detector!**

For  $M$  to be invertible, its determinant has to be non zero

- In practice,  $\det(M)$  should be non zero, which means that its eigenvalues should be large (i.e., not a flat region, not an edge) -> in practice, it **should be a corner or more generally contain texture!**

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$





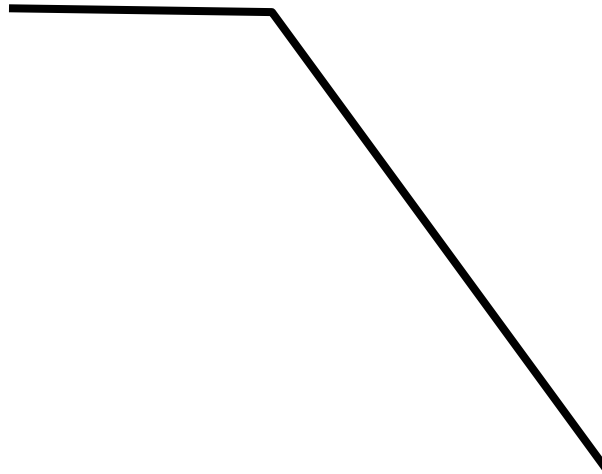
# Point Tracking

Edge – Low texture – High texture



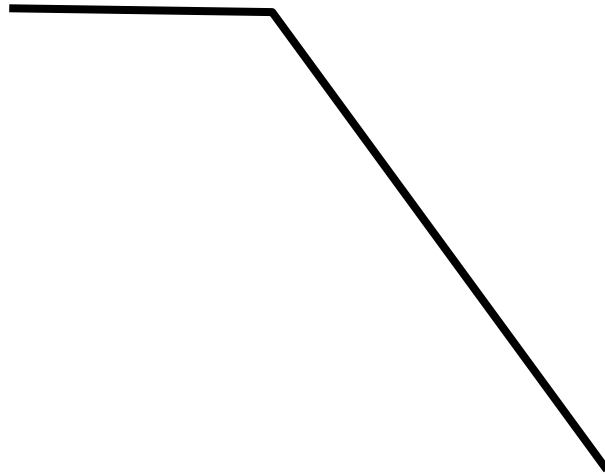
# Aperture Problem

- Consider the motion of the following corner



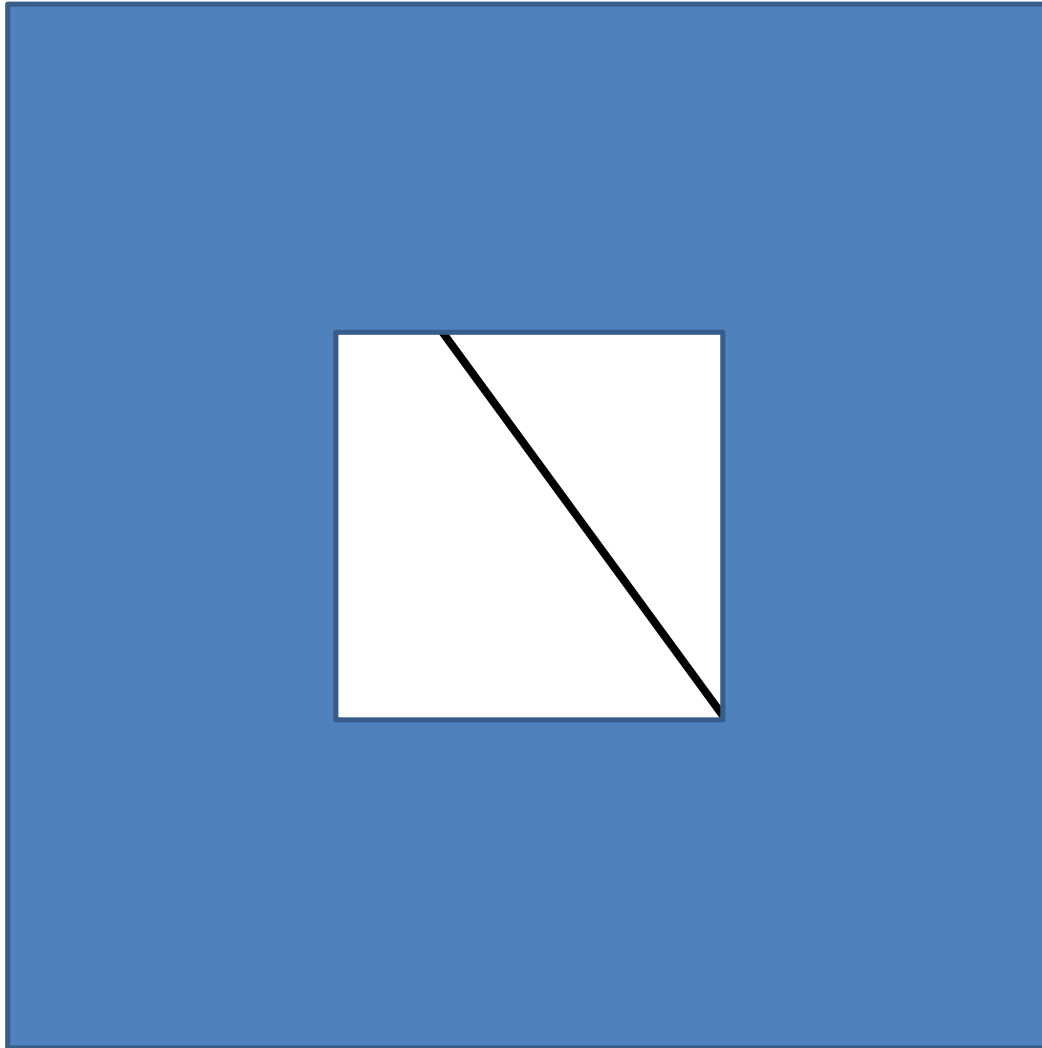
# Aperture Problem

- Consider the motion of the following corner



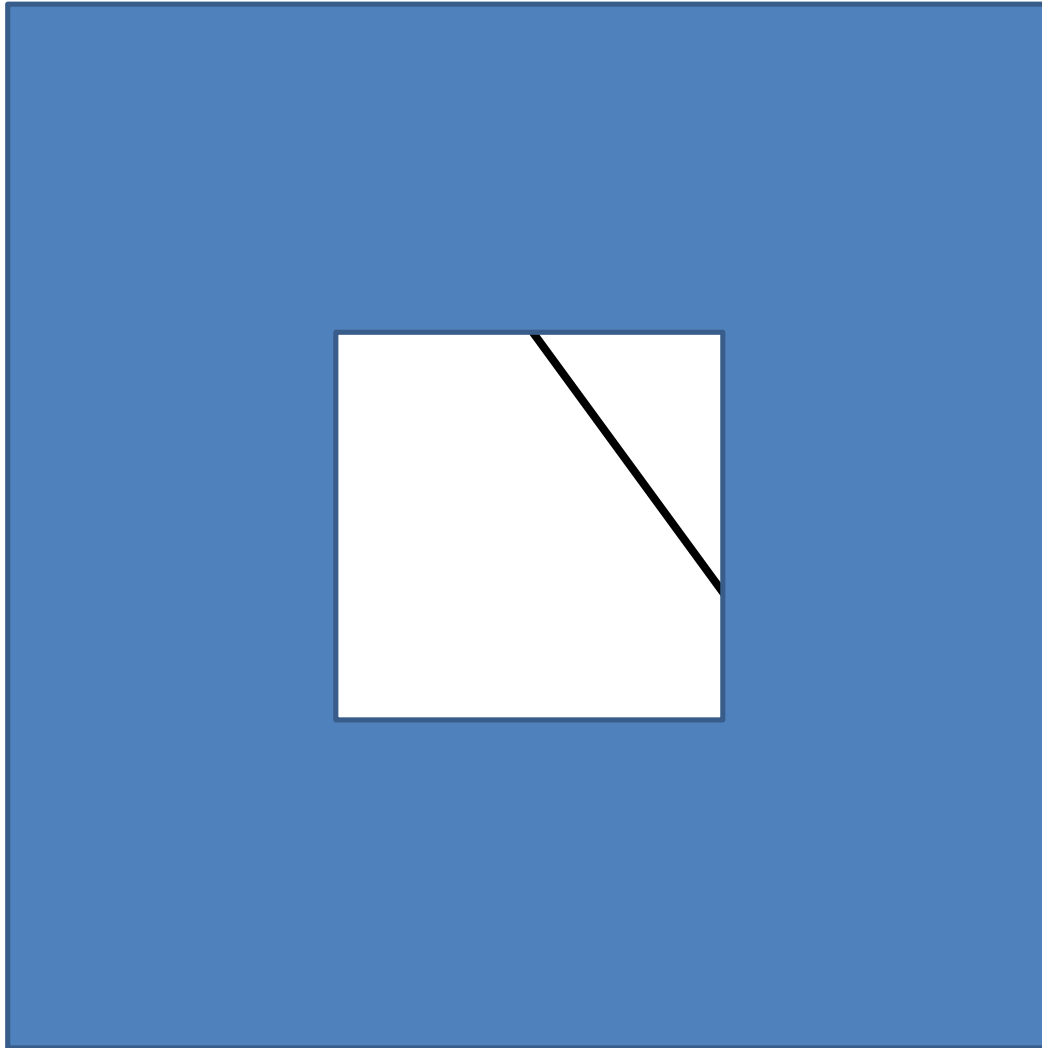
# Aperture Problem

- Now, look at the local brightness changes **through a small aperture**



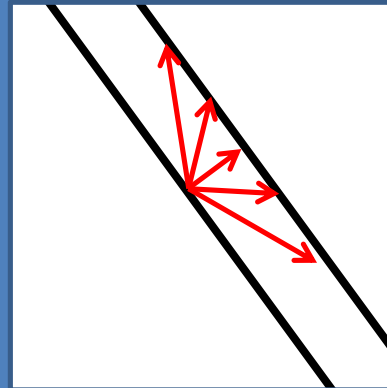
# Aperture Problem

- Now, look at the local brightness changes **through a small aperture**



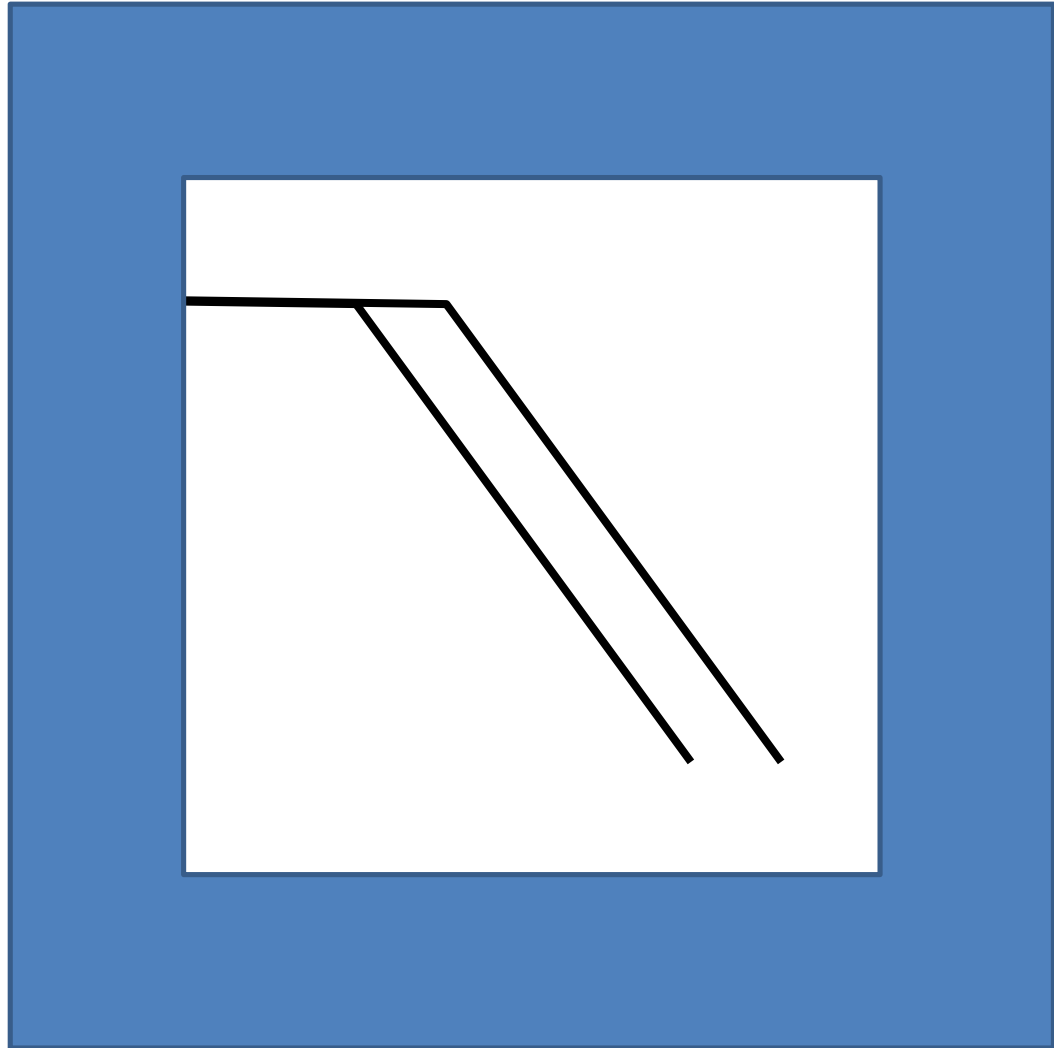
# Aperture Problem

- Now, look at the local brightness changes **through a small aperture**
- We **cannot always determine** the motion direction -> **Infinite motion solutions** may exist!
- **Solution?**



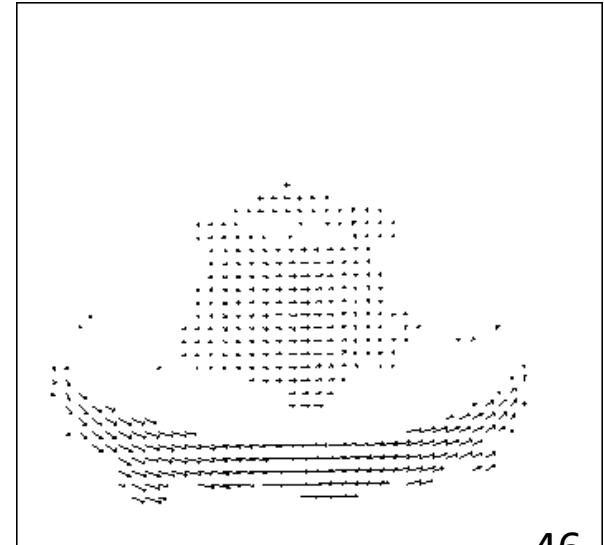
# Aperture Problem

- Now, look at the local brightness changes **through a small aperture**
- We **cannot always determine** the motion direction -> **Infinite motion solutions** may exist!
- **Solution?** Increase aperture size!



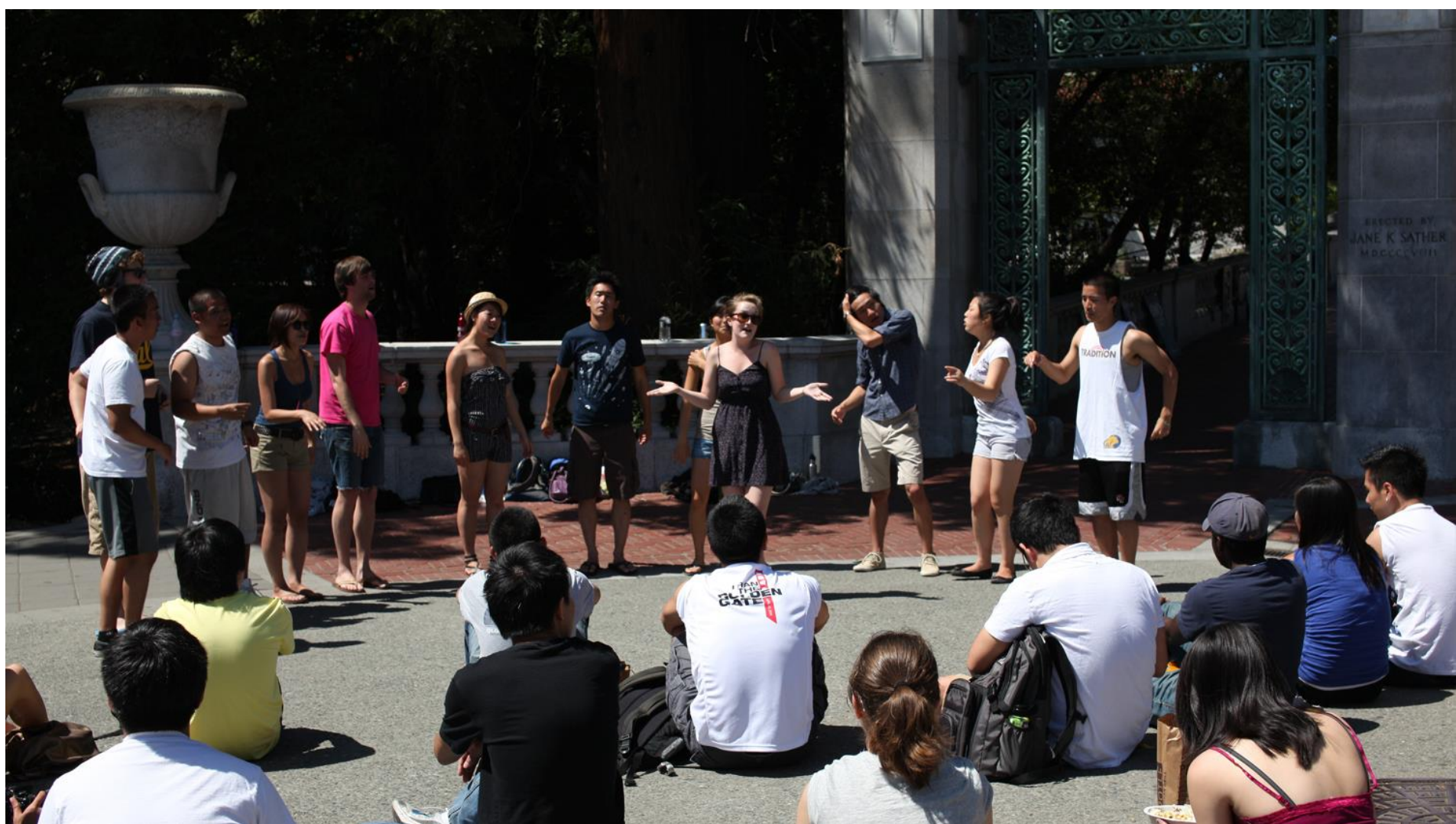
# Application of Differential Methods: Optical Flow

- **Optical flow** or **optic flow** is the pattern of apparent motion of objects in a visual scene caused by the relative motion between the observer (an eye or a camera) and the scene
- Tracks the motion of every pixels (or a grid of pixels) between two consecutive frames
- For each pixel, a **motion vector** is computed:
  - Vector **direction** represents motion direction
  - Vector **length** represents the amount of movement

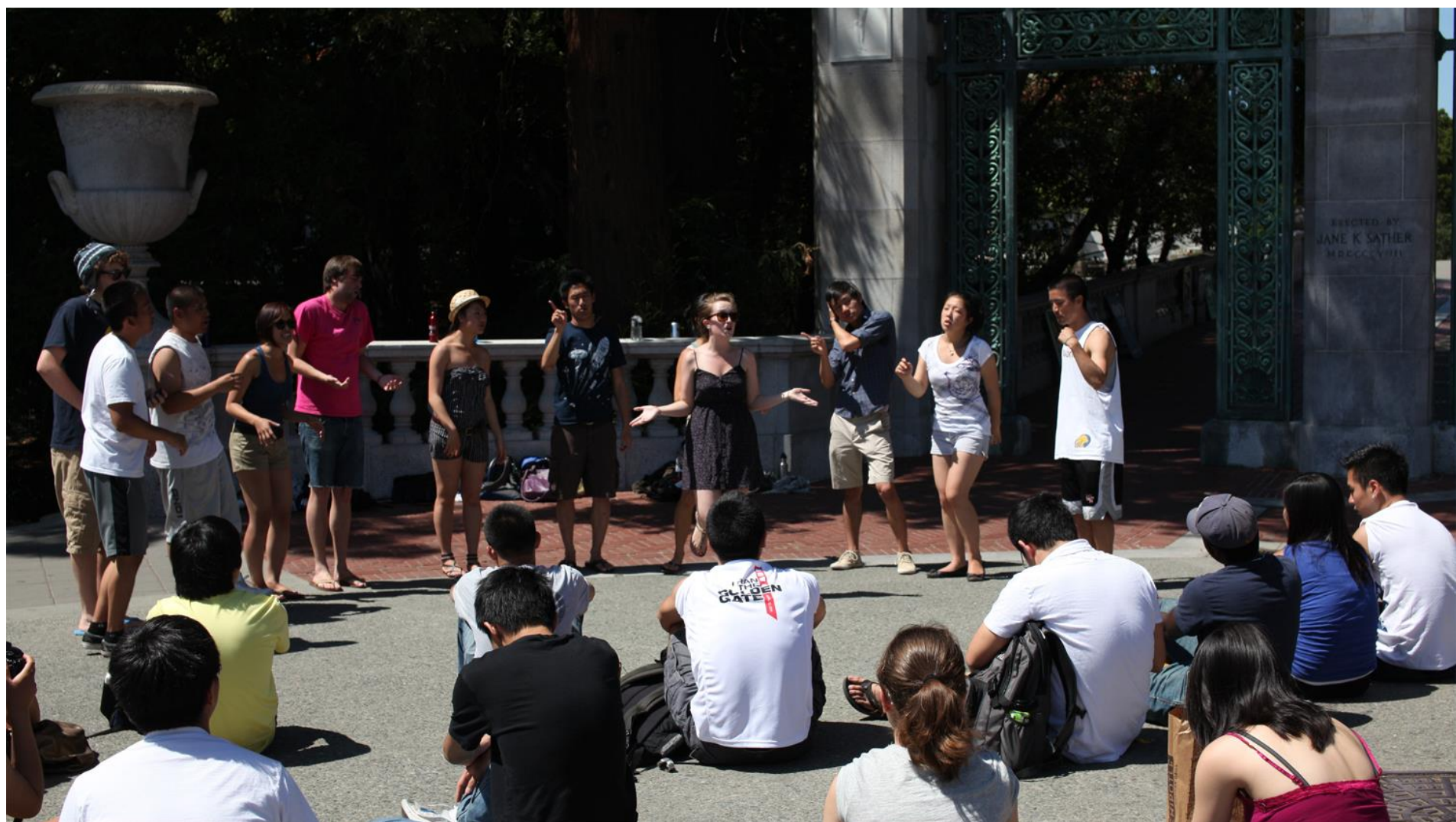




# Optical Flow

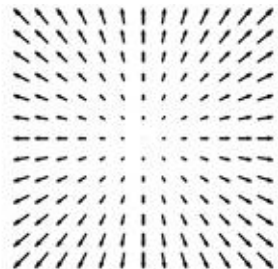
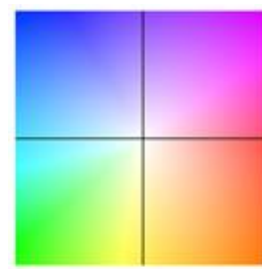


# Optical Flow





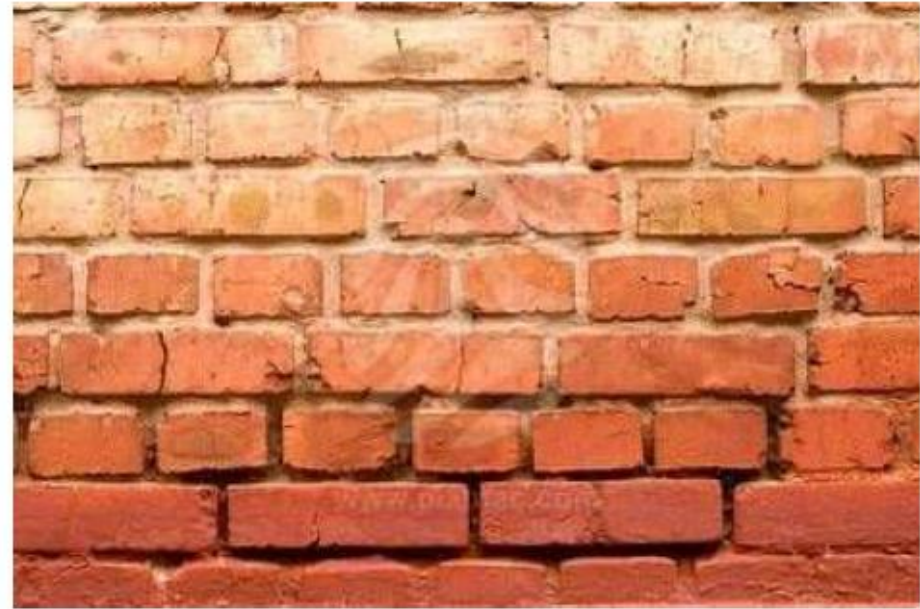
# Optical Flow



# Optical Flow example



# Optical flow issue: choosing the right patch size





# Application to Corner Tracking

Color encodes motion direction



# Block-based vs. Differential methods

- **Block-based methods:** search for the corresponding patch in a neighborhood of the point to be tracked. The search region is usually a square of  $n \times n$  pixels.



**Robust** to large motions



Can be **computationally expensive** ( $n \times n$  validations need to be made for a single point to track)

- **Differential methods:**



Works only for **small motions** (e.g., high frame rate). For larger motion, multi-scale implementations are used but are more expensive



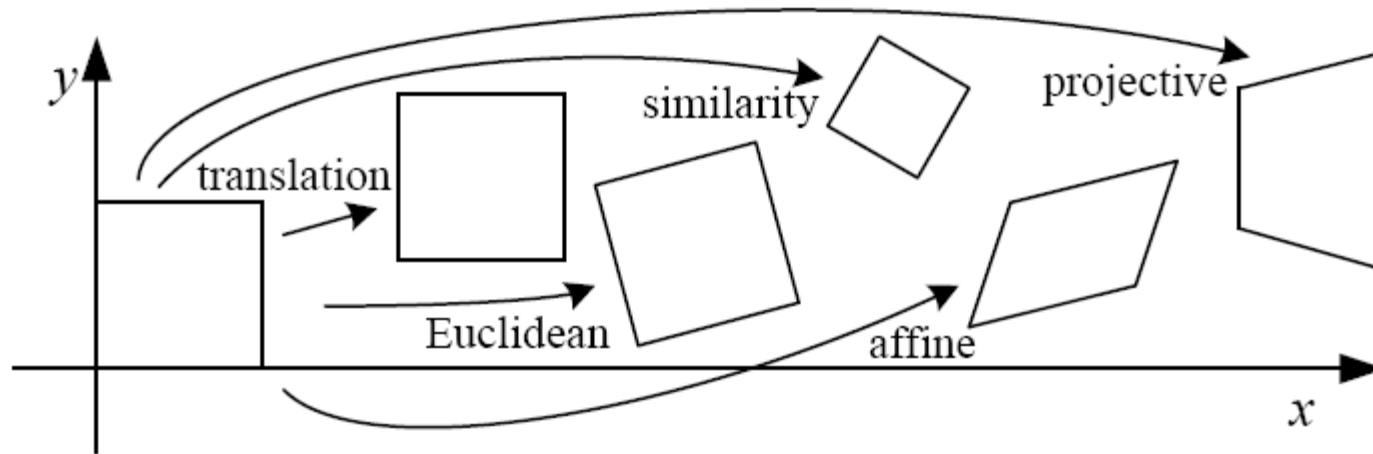
Much more **efficient** than block-based methods. Thus, can be used to track the motion of every pixel in the image (i.e., optical flow). It avoids searching in the neighborhood of the point by analyzing the **local intensity changes** (i.e., differences) of an image patch at a **specific location** (i.e., no search is performed).






# Outline

- What is Tracking?
- Point tracking
- Review of 2D image transformations and Jacobians
- Template tracking
- Tracking by detection of local image features



# Transformations – 2D



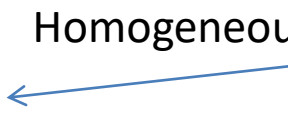
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I & t \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} R & t \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity	$\begin{bmatrix} sR & t \end{bmatrix}_{2 \times 3}$	4	angles + ...	
affine	$\begin{bmatrix} A \end{bmatrix}_{2 \times 3}$	6	parallelism + ...	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3 \times 3}$	8	straight lines	

# Summary of displacement models (2D transformations)






- Translation
$$\begin{aligned}x' &= x + a_1 \\ y' &= y + a_2\end{aligned}$$
- Euclidean
$$\begin{aligned}x' &= x\cos(a_3) - y\sin(a_3) + a_1 \\ y' &= x\sin(a_3) + y\cos(a_3) + a_2\end{aligned}$$
- Affine
$$\begin{aligned}x' &= a_1x + a_3y + a_5 \\ y' &= a_2x + a_4y + a_6\end{aligned}$$
- Projective  
(homography)
$$\begin{aligned}x' &= \frac{a_1x + a_2y + a_3}{a_7x + a_8y + 1} \\ y' &= \frac{a_4x + a_5y + a_6}{a_7x + a_8y + 1}\end{aligned}$$

# Summary of displacement models (2D transformations)

We call the transformation Warping  $W(\mathbf{x}, \mathbf{p})$  and  $\mathbf{p}$  the set of parameters  $p = (a_1, a_2, \dots, a_n)$

- Translation  $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} x + a_1 \\ y + a_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$   Homogeneous coordinates
- Euclidean  $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} x \cos(a_3) - y \sin(a_3) + a_1 \\ x \sin(a_3) + y \cos(a_3) + a_2 \end{bmatrix} = \begin{bmatrix} c a_3 & -s a_3 & a_1 \\ s a_3 & c a_3 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- Affine  $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} a_1 x + a_3 y + a_5 \\ a_2 x + a_4 y + a_6 \end{bmatrix} = \begin{bmatrix} a_1 & a_3 & a_5 \\ a_2 & a_4 & a_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- Projective  $W(\tilde{\mathbf{x}}, \mathbf{p}) = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

# Summary of displacement models (2D transformations)

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles + ...	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism + ...	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} ca_3 & -sa_3 & a_1 \\ sa_3 & ca_3 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x}, \mathbf{p}) = \lambda \begin{bmatrix} ca_3 & -sa_3 & a_1 \\ sa_3 & ca_3 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} a_1 & a_3 & a_5 \\ a_2 & a_4 & a_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$W(\tilde{\mathbf{x}}, \mathbf{p}) = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Derivative and gradient

- Function:  $f(x)$
- Derivative:  $f'(x) = \frac{df}{dx}$ , where  $x$  is a scalar
- Function:  $f(x_1, x_2, \dots, x_n)$
- Gradient:  $\nabla f(x_1, x_2, \dots, x_n) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

# Jacobian

- $F(x_1, x_2, \dots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{bmatrix}$

Vector-valued function

Derivative?

$$J(F) = \nabla F = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \dots, \frac{\partial f_1}{\partial x_n} \\ \vdots \\ \frac{\partial f_m}{\partial x_1}, \dots, \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$



Carl Gustav Jacob (1804-1851)

# Displacement-model Jacobians $\nabla W_p$

$$p = (a_1, a_2, \dots, a_n)$$

- Translation

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} x + a_1 \\ y + a_2 \end{bmatrix} \quad \nabla W_p = \begin{bmatrix} \frac{\partial W_1}{\partial a_1} & \frac{\partial W_1}{\partial a_2} \\ \frac{\partial W_2}{\partial a_1} & \frac{\partial W_2}{\partial a_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Euclidean

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} x \cos(a_3) - y \sin(a_3) + a_1 \\ x \sin(a_3) + y \cos(a_3) + a_2 \end{bmatrix} \quad \nabla W_p = \begin{bmatrix} 1 & 0 & -x \sin(a_3) - y \cos(a_3) \\ 0 & 1 & x \cos(a_3) - y \sin(a_3) \end{bmatrix}$$

- Affine

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} a_1 x + a_3 y + a_5 \\ a_2 x + a_4 y + a_6 \end{bmatrix} \quad \nabla W_p = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

# Outline

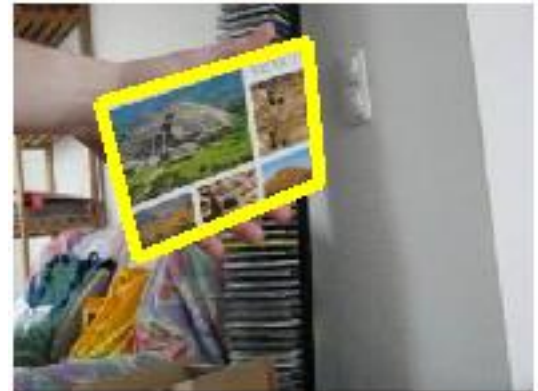
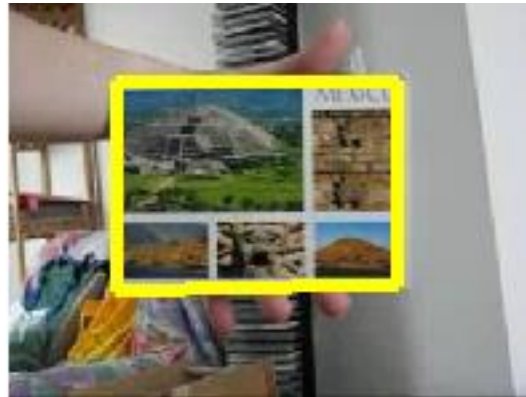
- What is Tracking?
- Point tracking
- Review of 2D image transformations and Jacobians
- Template tracking: Lucas-Kanade algorithm
- Tracking by detection of local image features



# Template tracking

Definition: follow a template image in a video sequence by estimating the warp

**Template image**

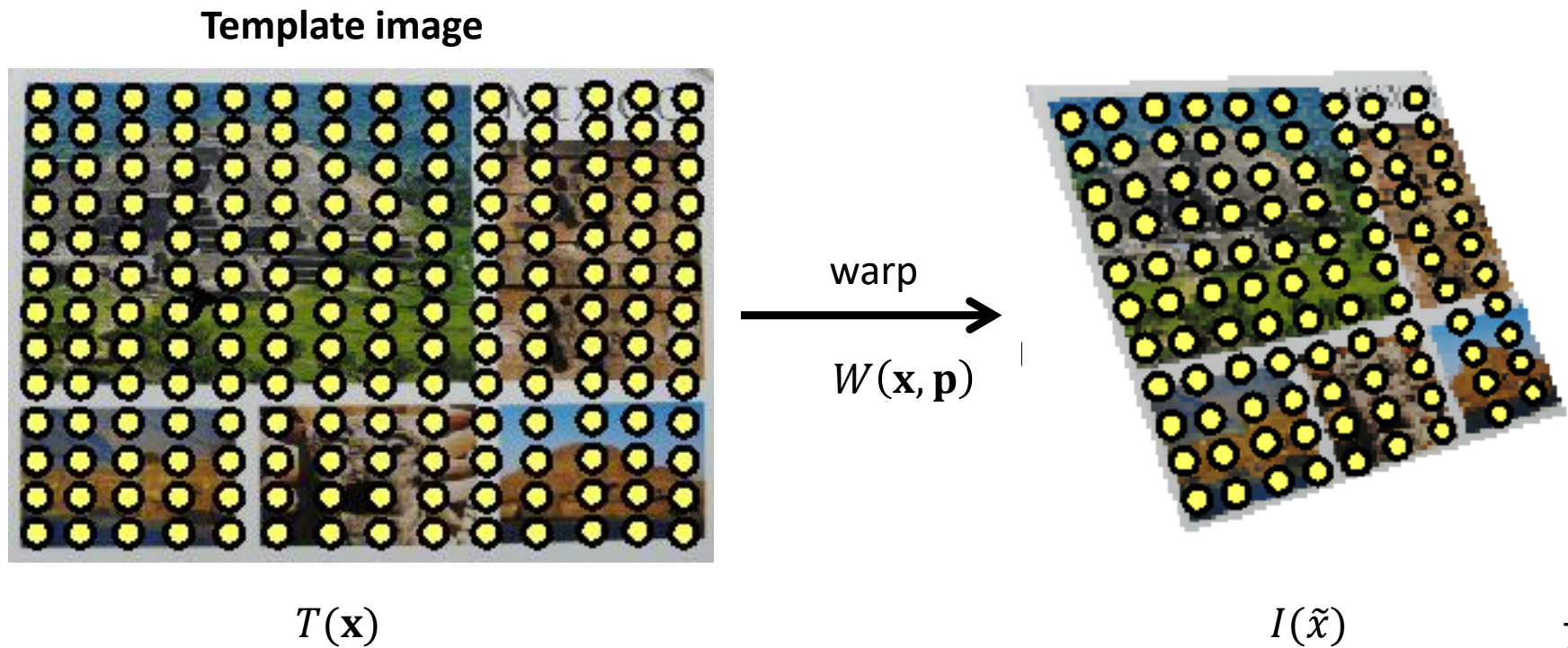


# The Lucas-Kanade tracker



# Template warping

- Given the template image  $T(\mathbf{x})$
- Take all pixels from the template image  $T(\mathbf{x})$  and warp them using the function  $W(\mathbf{x}, \mathbf{p})$  parameterized in terms of parameters  $\mathbf{p}$



# Template Tracking: Problem Formulation

- The goal of template-based tracking is to find the set of warp parameters  $\mathbf{p}$  such that:

$$I(W(\mathbf{x}, \mathbf{p})) = T(\mathbf{x})$$

This is solved by determining  $\mathbf{p}$  that minimizes the Sum of Squared Differences

$$E = SSD = \sum_{\mathbf{x} \in \mathbf{T}} [I(W(\mathbf{x}, \mathbf{p})) - T(\mathbf{x})]^2$$

# Assumptions

- **No errors in the template image boundaries:** only the appearance of the object to be tracked appears in the template image
- **No occlusion:** the entire template is visible in the input image
- **Brightness constancy assumption:** the intensity of the object appearance is always the same across different views



# The Lucas-Kanade tracker

- Uses the Gauss-Newton method for minimization, that is:
  - Applies a first-order approximation of the warp
  - Attempts to minimize the SSD iteratively

# Derivation of the Lucas-Kanade algorithm

$$E = \sum_{\mathbf{x} \in \mathbf{T}} [I(W(\mathbf{x}, \mathbf{p})) - T(\mathbf{x})]^2$$

How do I get the initial estimate?

- Assume that an initial estimate of  $\mathbf{p}$  is known. Then, we want to find the increment  $\Delta \mathbf{p}$  that minimizes

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

- First-order Taylor approximation of  $I(W(\mathbf{x}, \mathbf{p} + \Delta \mathbf{p}))$  yields to:

$$I(W(\mathbf{x}, \mathbf{p} + \Delta \mathbf{p})) \cong I(W(\mathbf{x}, \mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta \mathbf{p}$$

$\nabla I = [I_x, I_y]$  = Image gradient evaluated at  $W(\mathbf{x}, \mathbf{p})$       Jacobian of the warp  $W(\mathbf{x}, \mathbf{p})$

# Derivation of the Lucas-Kanade algorithm

$$E = \sum_{\mathbf{x} \in \mathbf{T}} \left[ I(W(\mathbf{x}, \mathbf{p} + \Delta \mathbf{p})) - T(\mathbf{x}) \right]^2$$

- By replacing  $I(W(\mathbf{x}, \mathbf{p} + \Delta \mathbf{p}))$  with its 1<sup>st</sup> order approximation, we get

$$E = \sum_{\mathbf{x} \in \mathbf{T}} \left[ I(W(\mathbf{x}, \mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

- How do we minimize it?
- We differentiate  $E$  with respect to  $\Delta \mathbf{p}$  and we equate it to zero, i.e.,

$$\frac{\partial E}{\partial \Delta \mathbf{p}} = 0$$



# Derivation of the Lucas-Kanade algorithm

$$E = \sum_{\mathbf{x} \in \mathbf{T}} \left[ I(W(\mathbf{x}, \mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

$$\frac{\partial E}{\partial \Delta \mathbf{p}} = 2 \sum_{\mathbf{x} \in \mathbf{T}} \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^T \left[ I(W(\mathbf{x}, \mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]$$

$$\frac{\partial E}{\partial \Delta \mathbf{p}} = 0$$

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

# Derivation of the Lucas-Kanade algorithm

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

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

Second moment matrix (Hessian) of the warped image

**What does H look like when the warp is a pure translation?**

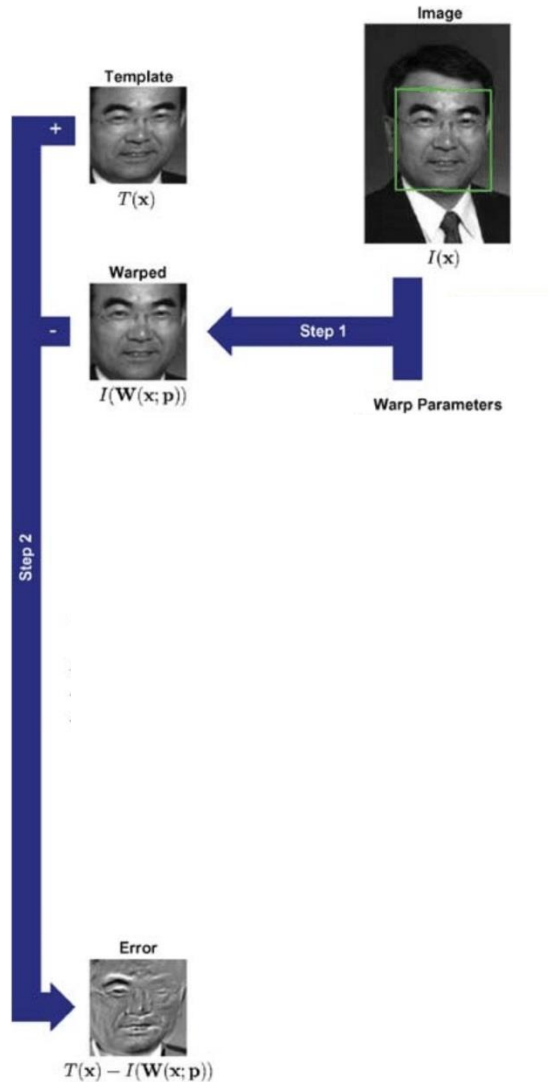
# Lucas-Kanade algorithm

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

1. Warp  $I(\mathbf{x})$  with  $W(\mathbf{x}, \mathbf{p}) \rightarrow I(W(\mathbf{x}, \mathbf{p}))$
2. Compute the error: subtract  $I(W(\mathbf{x}, \mathbf{p}))$  from  $T(\mathbf{x})$
3. Compute **warped** gradients:  $\nabla I = [I_x, I_y]$ , evaluated at  $W(\mathbf{x}, \mathbf{p})$
4. Evaluate the Jacobian of the warping:  $\frac{\partial W}{\partial \mathbf{p}}$
5. Compute steepest descent:  $\nabla I \frac{\partial W}{\partial \mathbf{p}}$
6. Compute Inverse Hessian:  $H^{-1} = \left[ \sum_{\mathbf{x} \in \mathbf{T}} \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^T \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right] \right]^{-1}$
7. Multiply steepest descent with error:  $\sum_{\mathbf{x} \in \mathbf{T}} \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^T [T(\mathbf{x}) - I(W(\mathbf{x}, \mathbf{p}))]$
8. Compute  $\Delta \mathbf{p}$
9. Update parameters:  $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$
10. Repeat until  $\Delta \mathbf{p} < \varepsilon$

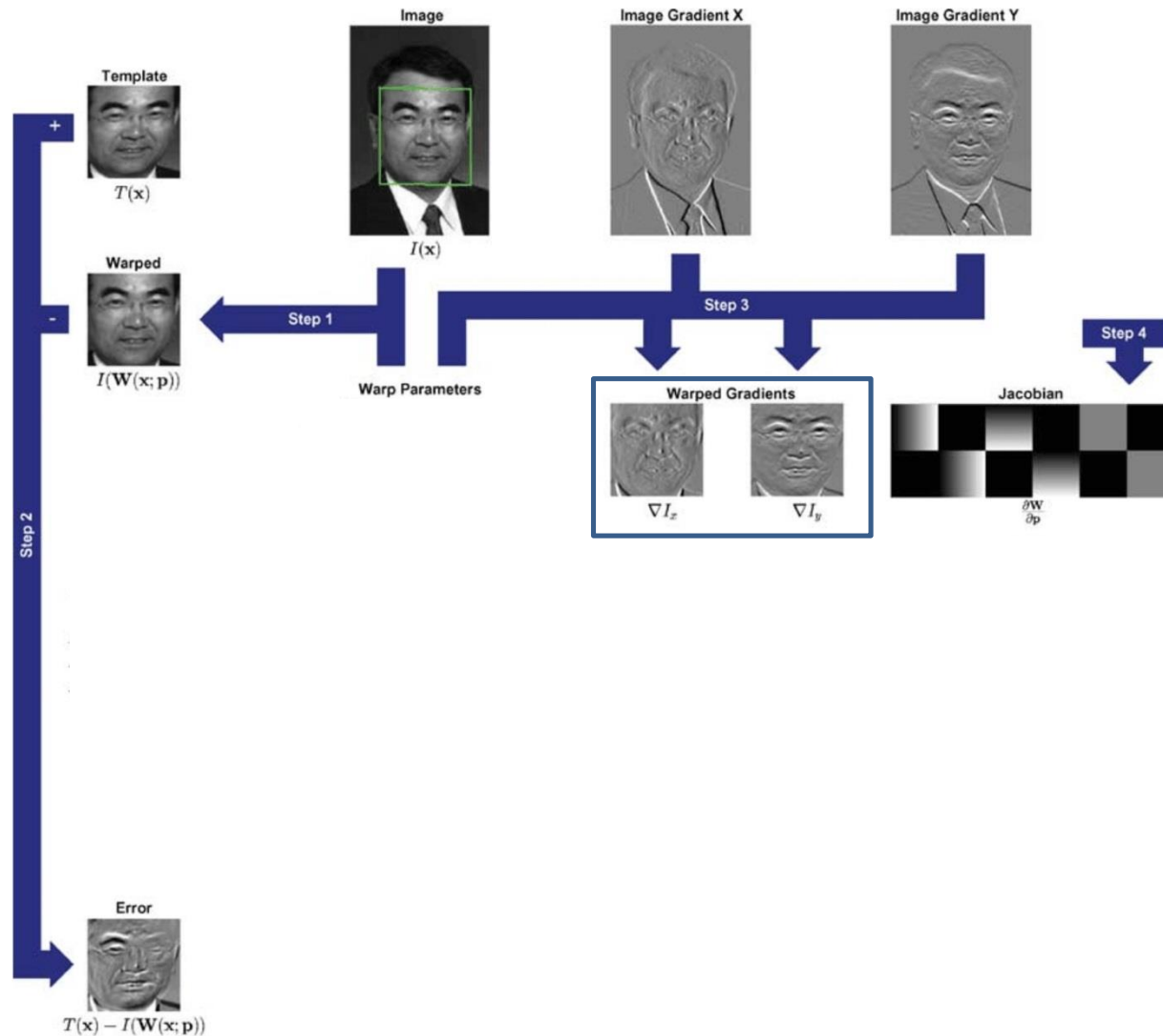
# Lucas-Kanade algorithm

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



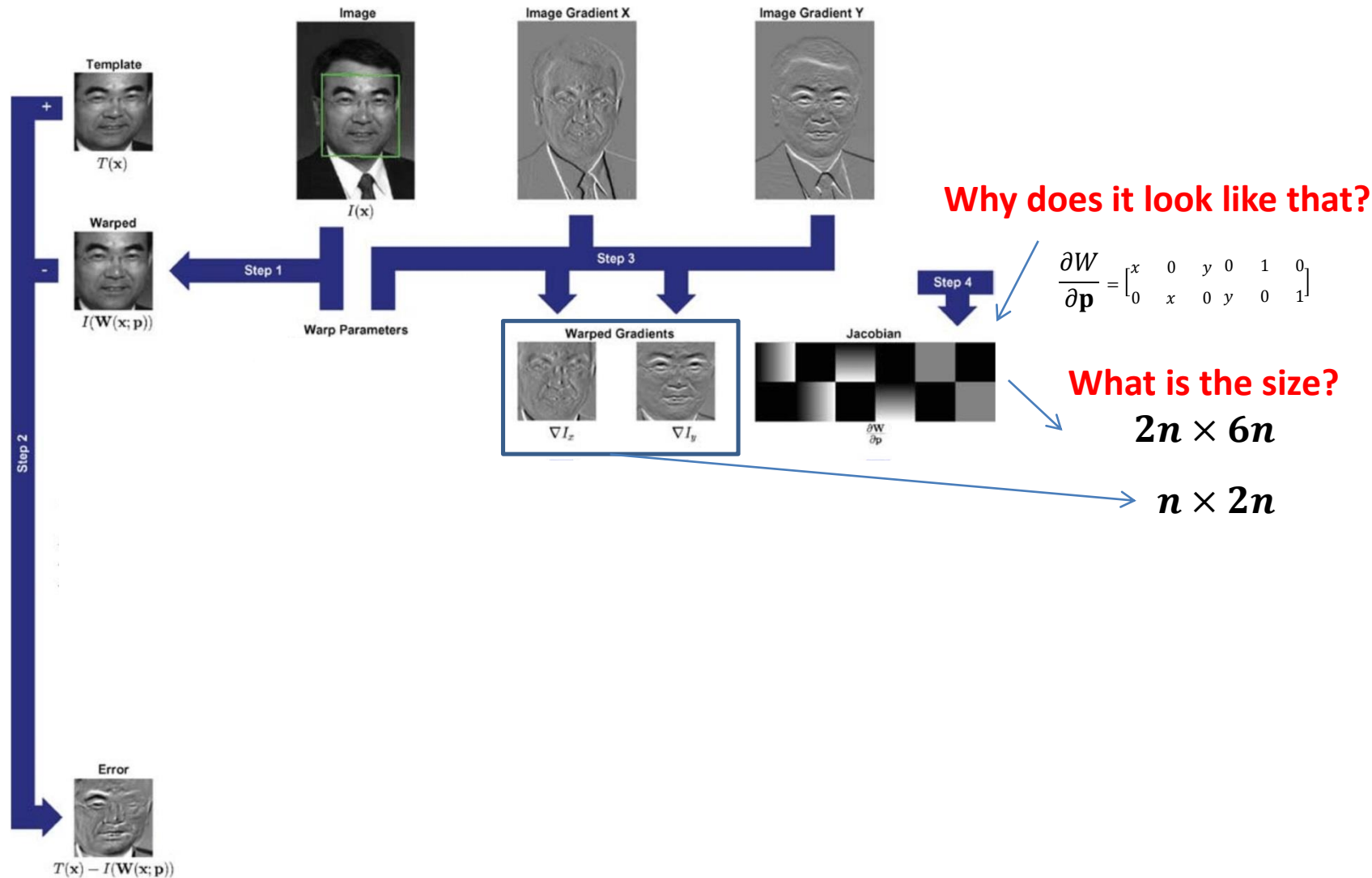
# Lucas-Kanade algorithm

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



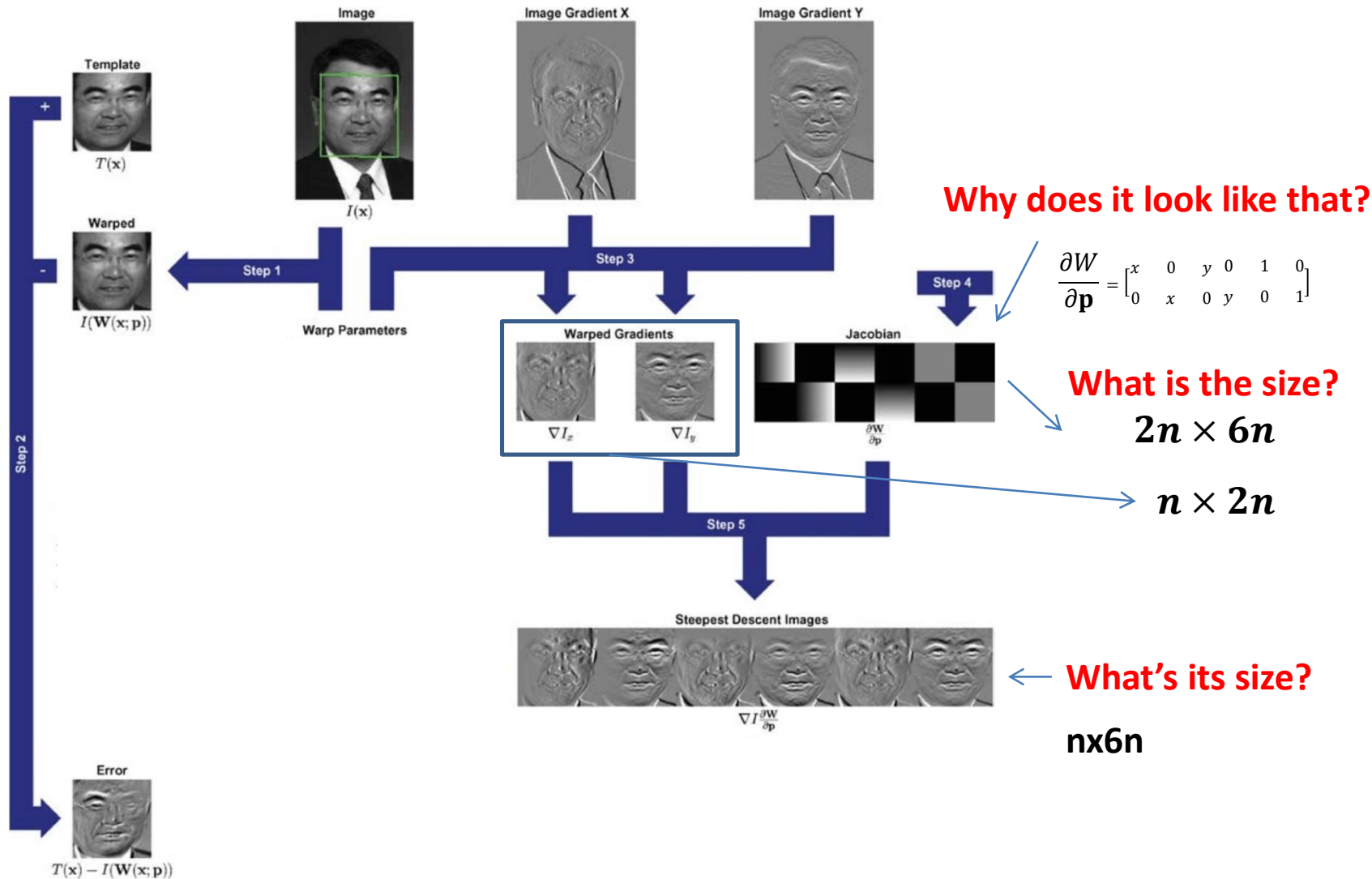
# Lucas-Kanade algorithm

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



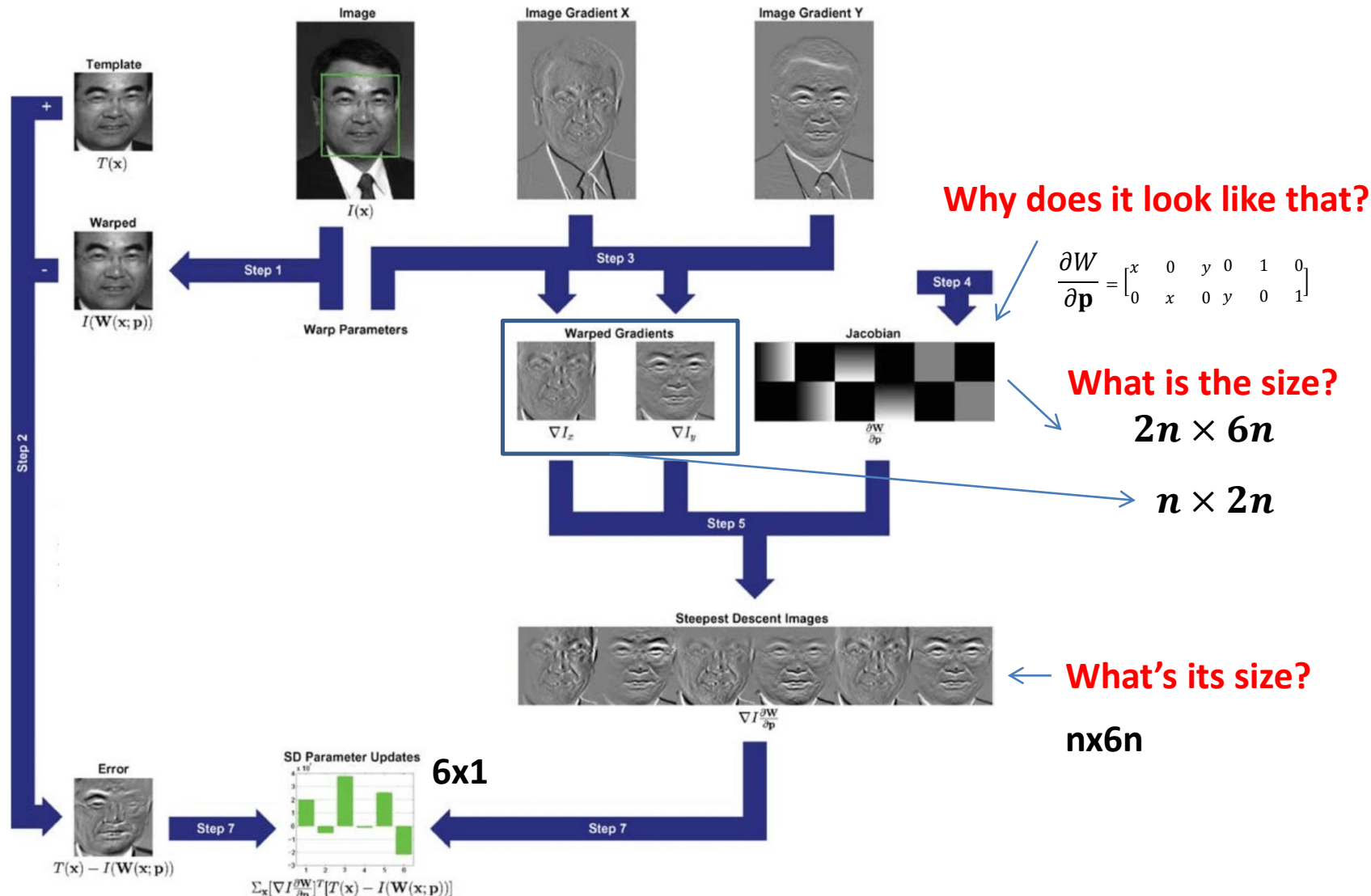
# Lucas-Kanade algorithm

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



# Lucas-Kanade algorithm

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





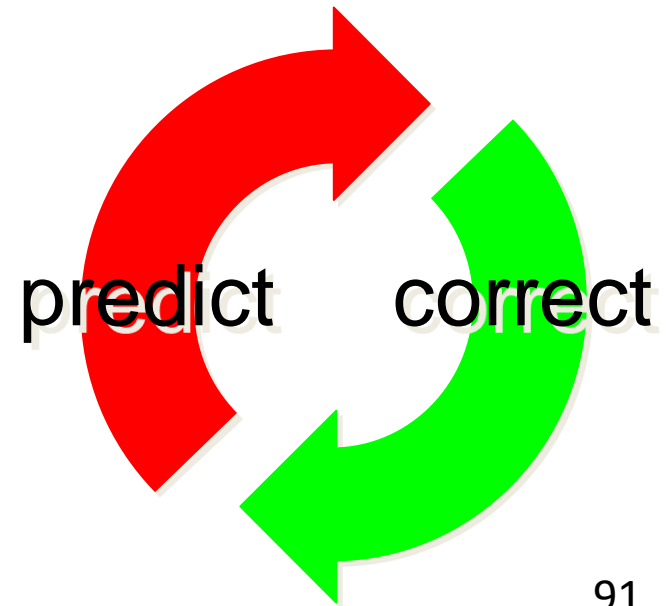
$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x} \in \mathbf{T}} \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^T [T(\mathbf{x}) - I(W(\mathbf{x}, \mathbf{p}))]$$


89

# Lucas-Kanade algorithm: Discussion

Lucas-Kanade follows a *predict-correct* cycle

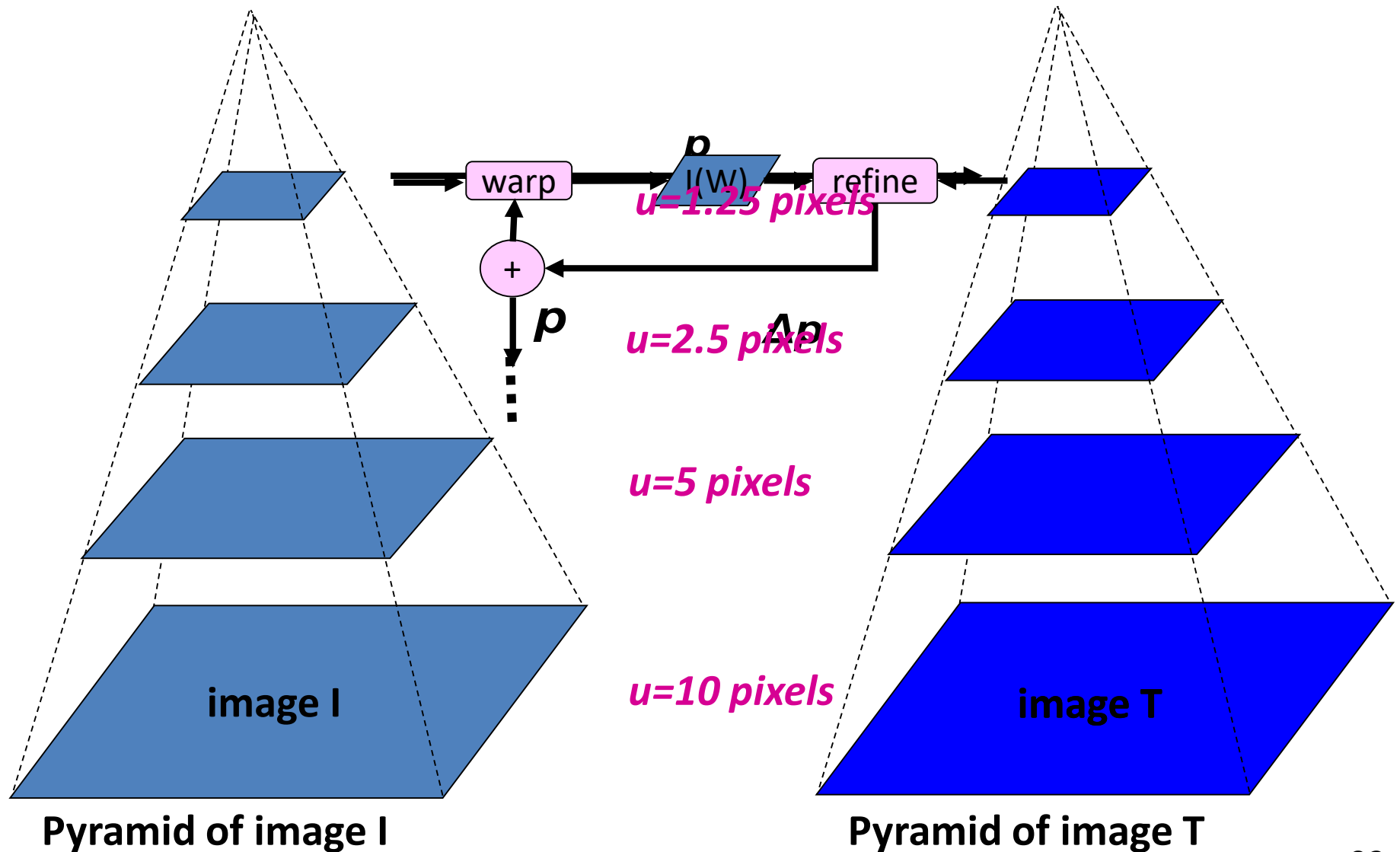
- A *prediction*  $I(W(\mathbf{x}, \mathbf{p}))$  of the warped image is computed from an initial estimate
- The *correction* parameter  $\Delta \mathbf{p}$  is computed as a function of the error  $T(\mathbf{x}) - I(W(\mathbf{x}, \mathbf{p}))$  between the prediction and the template
- The larger this error, the larger the correction applied



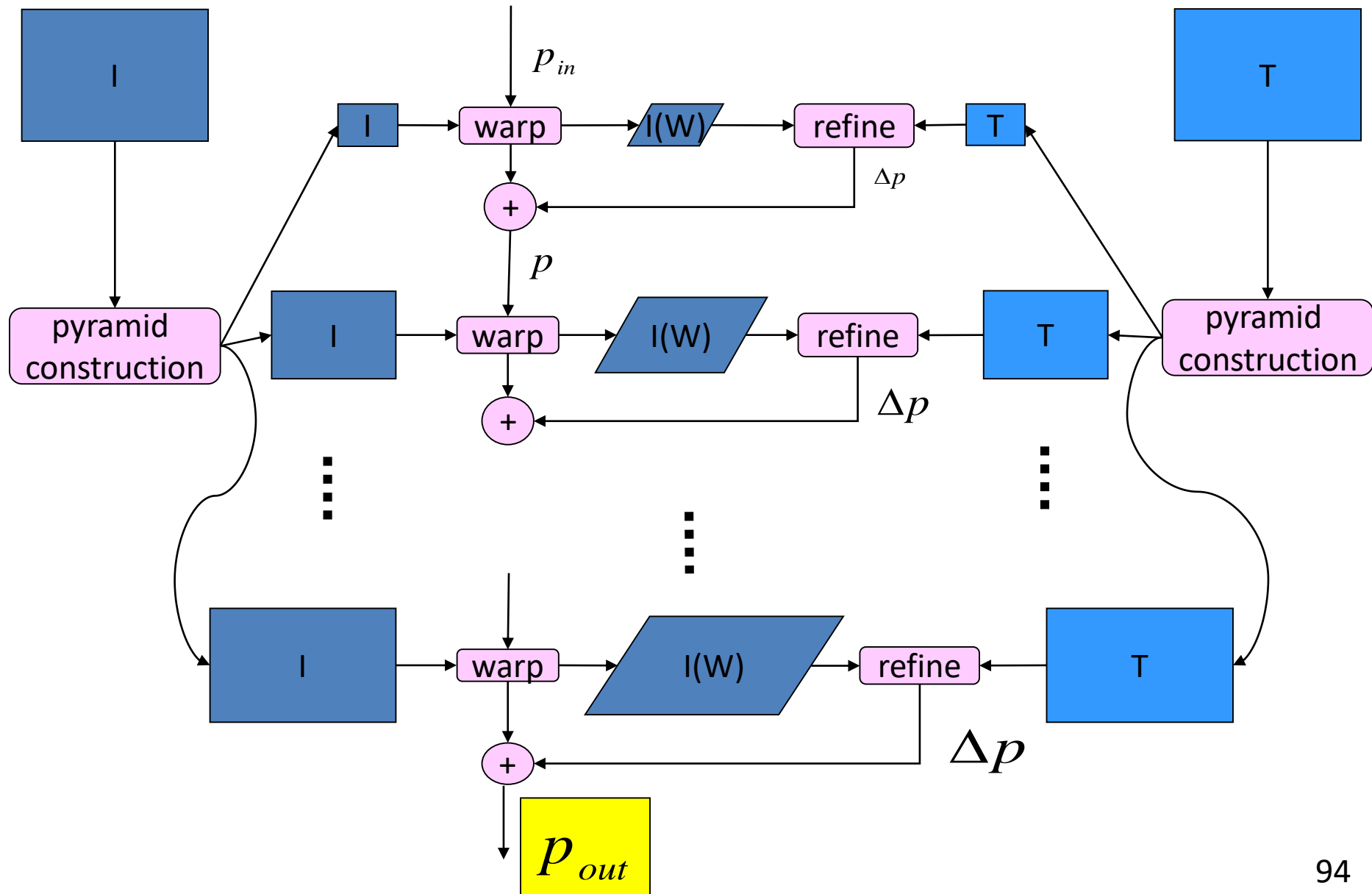
# Lucas-Kanade algorithm: Discussion

- How to get the initial estimate  $\mathbf{p}$ ?
- When does the Lucas-Kanade fail?
  - If the initial estimate is too far, then the linear approximation does not longer hold -> solution?
    - Pyramidal implementations (see next slide)
- Other problems:
  - Deviations from the mathematical model: object deformations, illumination changes, etc.
  - Occlusions
  - Due to these reasons, tracking may drift -> solution?
    - Update the template with the last image

# Coarse-to-fine estimation



# Coarse-to-fine estimation



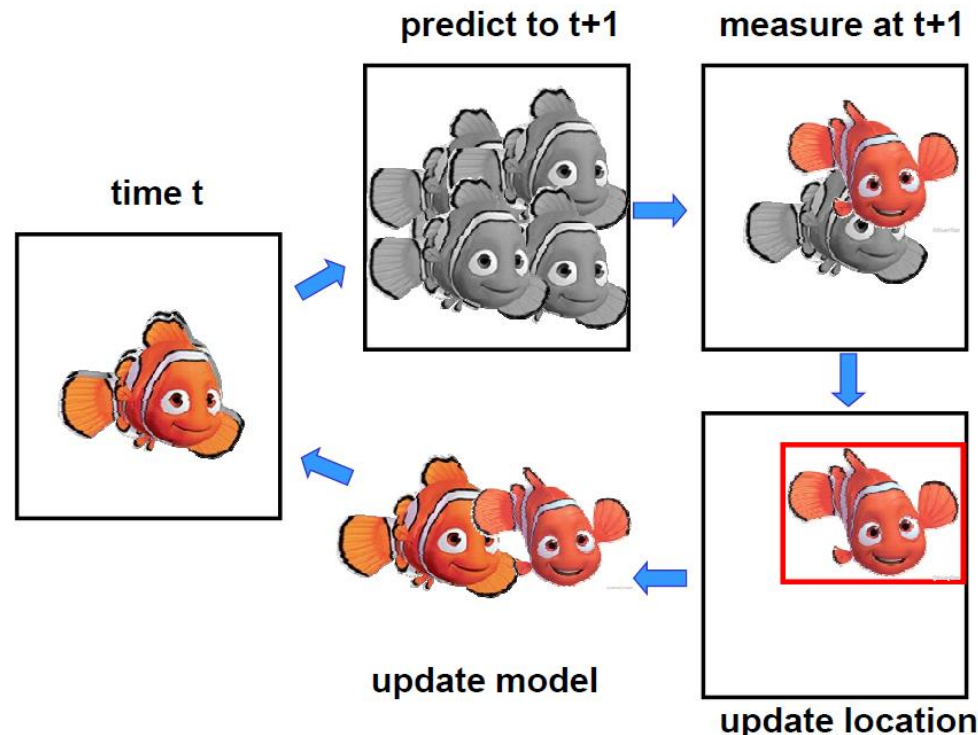
# Generalization of Lucas-Kanade

- The same concept (predict/correct) can be applied to tracking of 3D object (in this case, what is the transformation to estimate? What is the template?)



# Generalization of Lucas-Kanade

- The same concept (predict/correct) can be applied to tracking of 3D object (**in this case, what is the transformation to estimate? What is the template?**)
- In order to deal with wrong prediction, it can be implemented in a **Particle-Filter** fashion (using multiple hypotheses that need to be validated)



# Outline

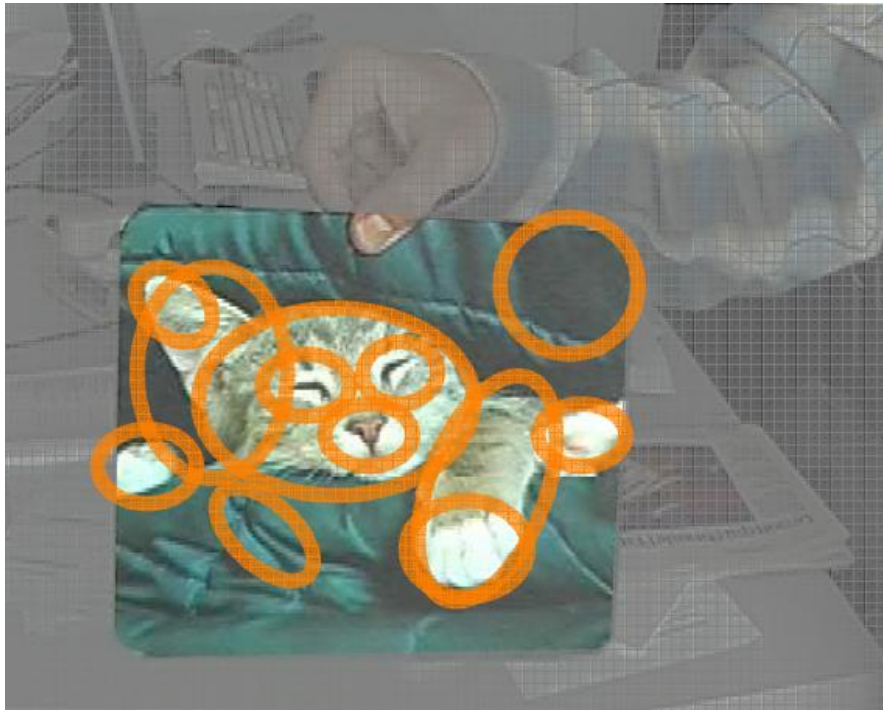
- Point tracking
- Template tracking
- Tracking by detection of local image features



# Tracking by detection of local image features

## Step 1: Keypoint detection and matching

- invariant to scale, rotation, or perspective



Template image with the object to detect

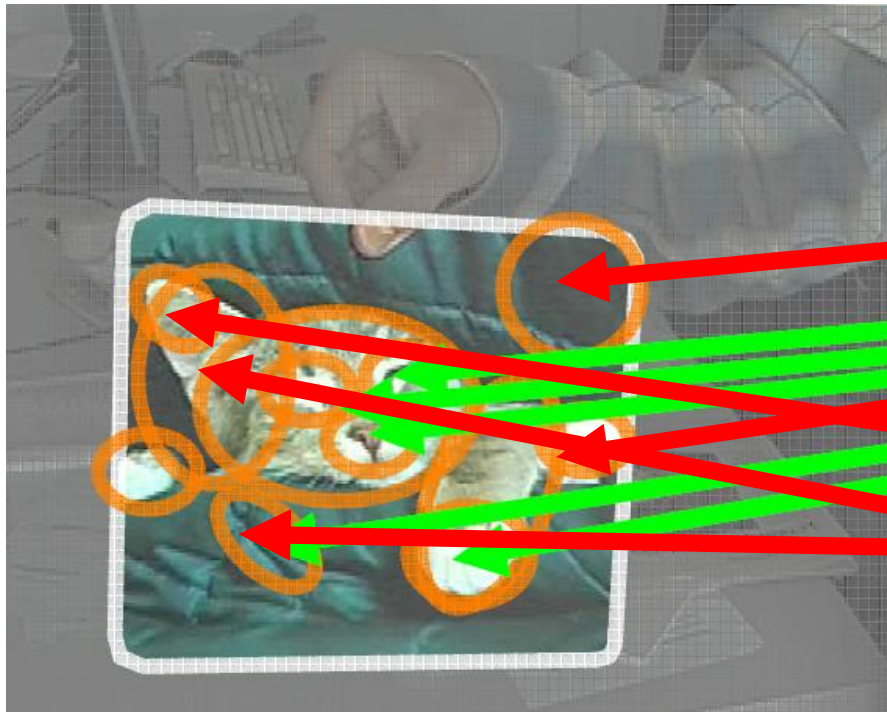


Current test image

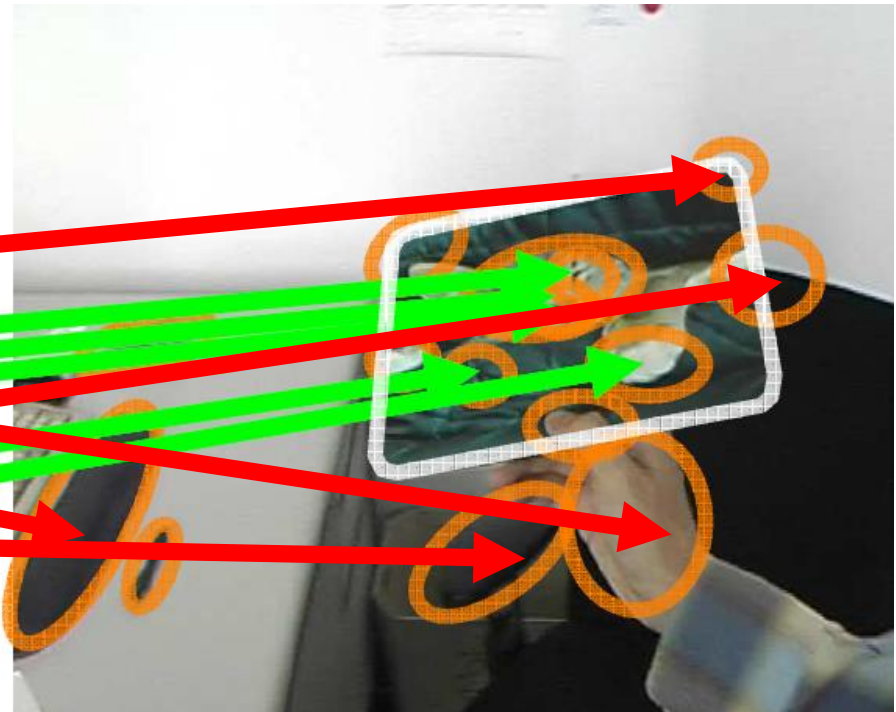
# Tracking by detection of local image features

## Step 1: Keypoint detection and matching

- invariant to scale, rotation, or perspective



Template image with the object to detect



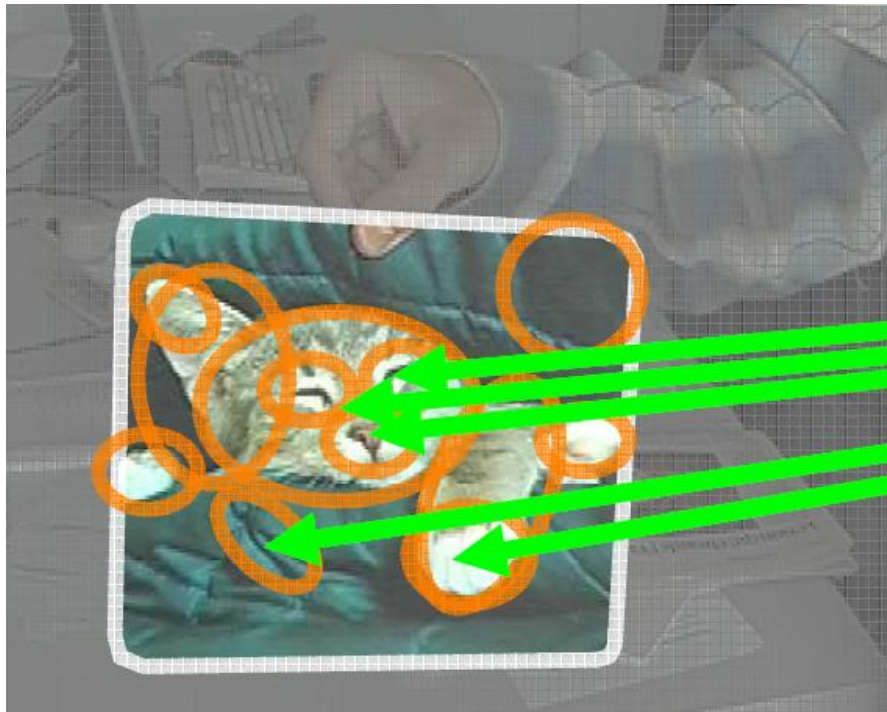
Current test image

# Tracking by detection of local image features

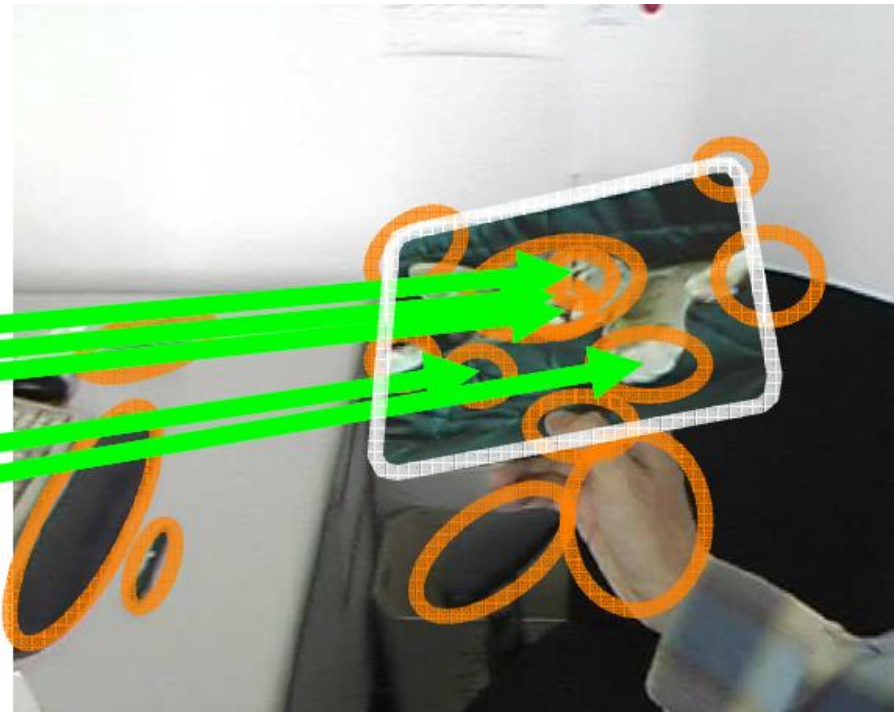
Step 1: Keypoint detection and matching

- invariant to scale, rotation, or perspective

Step 2: Geometric verification (RANSAC)



Template image with the object to detect



Current test image

# Tracking by detection of local image features





# Tracking issues

- How to segment the object to track from background?
- How to initialize the warping?
- How to handle occlusions
- How to handle illumination changes and non modeled effects?



# References

- Chapter 8 of Szeliski's book
- **Original paper:** B. D. Lucas and T. Kanade (1981), [An iterative image registration technique with an application to stereo vision.](#) Proceedings of Imaging Understanding Workshop, pages 121—130
- Baker, Matthews, “[Lucas-Kanade 20 Years On: A Unifying Framework](#)”, IJCV, 2004.

# Implementations

- **OpenCV implementation :**

[http://docs.opencv.org/modules/video/doc/motion\\_analysis\\_and\\_object\\_tracking.html?highlight=lucas%20kanade](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=lucas%20kanade)

- **Some Matlab implementations:**

- 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>

[http://vision.eecs.ucf.edu/Code/Optical\\_Flow/Lucas%20Kanade.zip](http://vision.eecs.ucf.edu/Code/Optical_Flow/Lucas%20Kanade.zip)