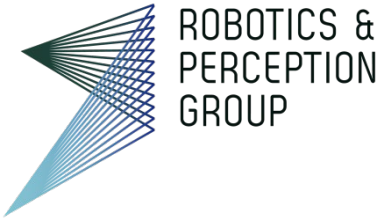
University of Zurich[UZH]

ETH zürich



Vision Algorithms for Mobile Robotics

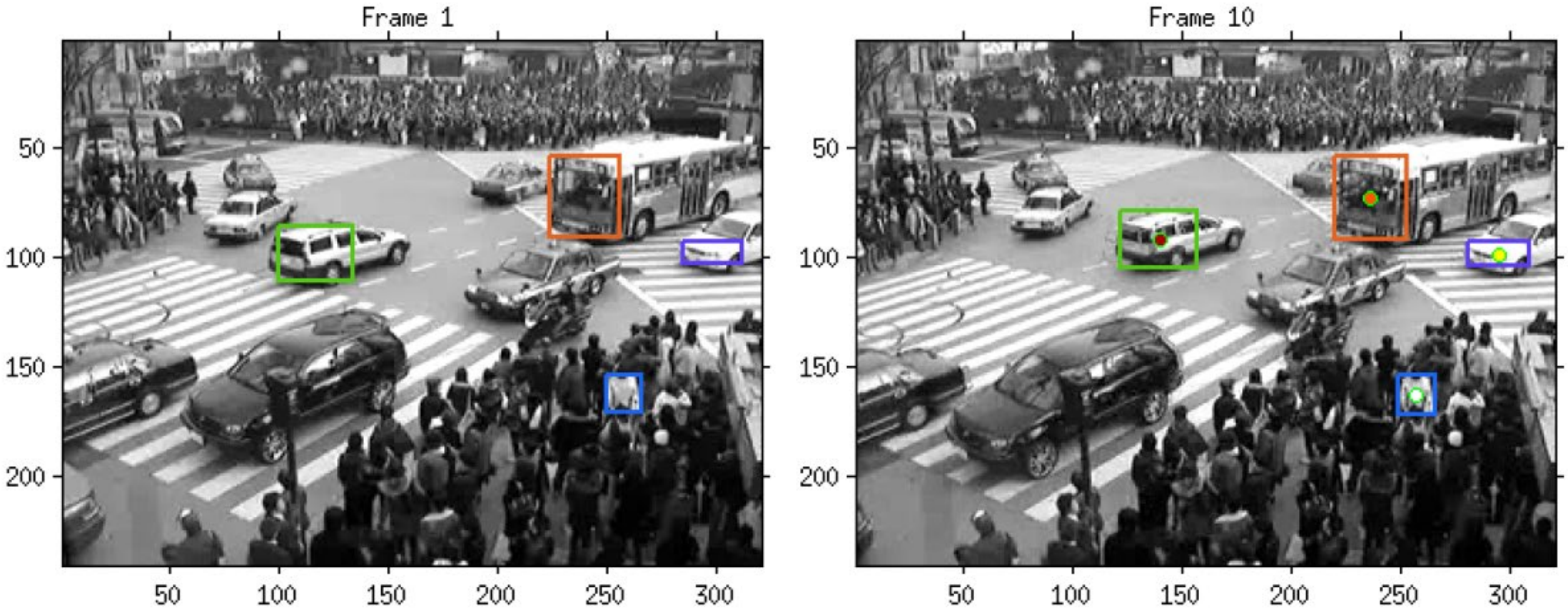
Lecture 11
Tracking

Davide Scaramuzza

http://rpg.ifi.uzh.ch

Lab Exercise – This afternoon

Implement the Kanade-Lucas-Tomasi (KLT) tracker

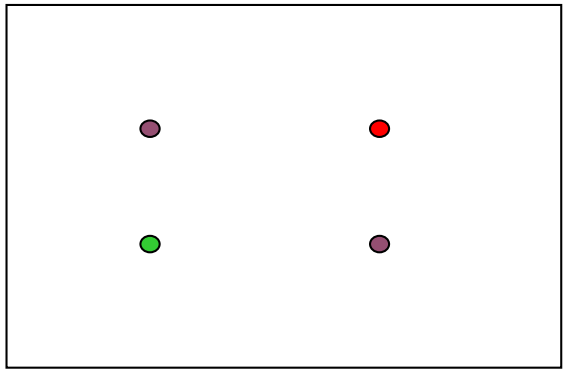


Outline

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

Point Tracking

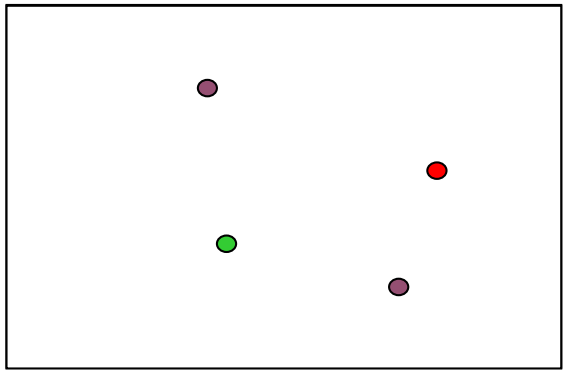
- **Problem:** given two images, estimate the motion of a pixel point from image I_0 to image I_1



$I_0(x, y)$

Point Tracking

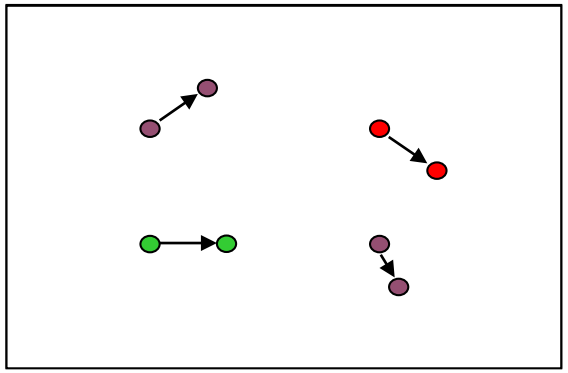
- **Problem:** given two images, estimate the motion of a pixel point from image I_0 to image I_1



$I_1(x, y)$

Point Tracking

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



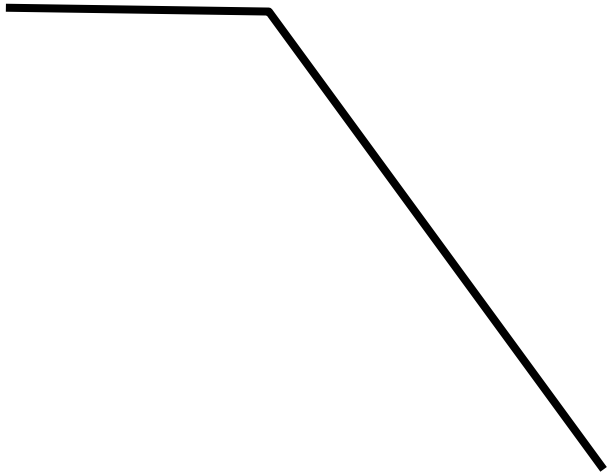
$I_0(x, y)$

(u, v): optical flow vector

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

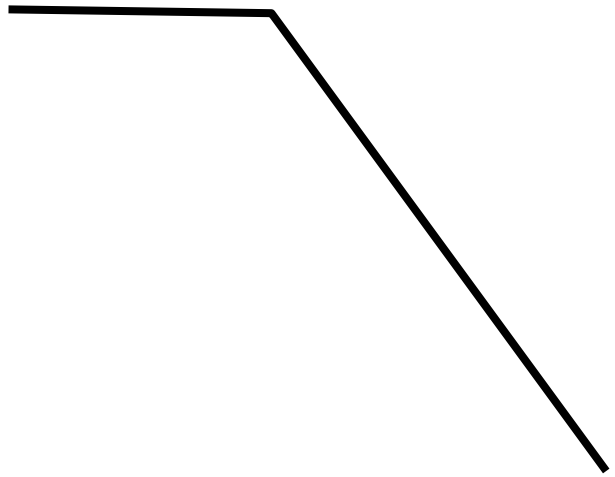
Point Tracking

- Consider the motion of the following corner



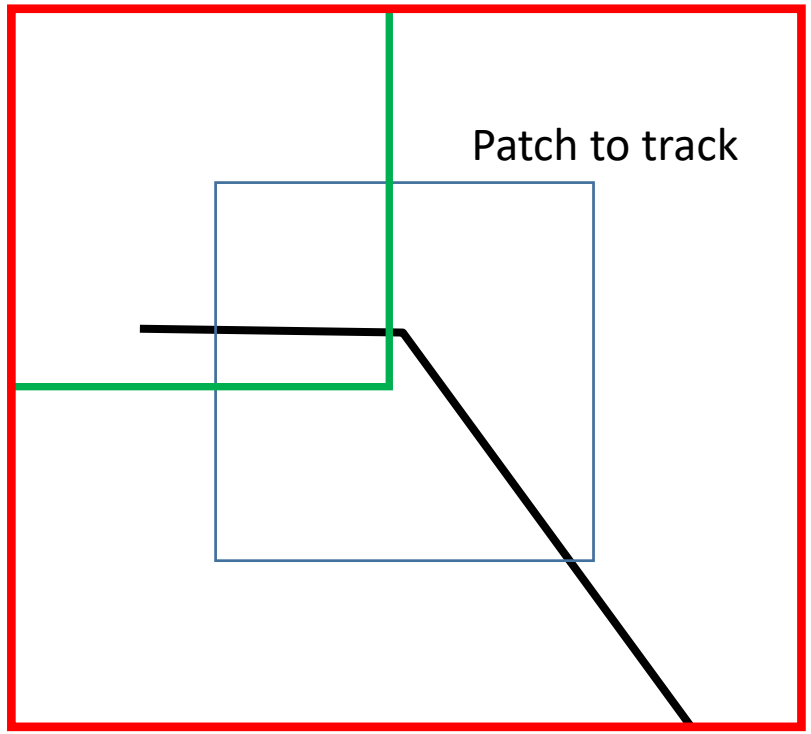
Point Tracking

- Consider the motion of the following corner



Point Tracking with Block Matching

- Search for the corresponding patch in a $D \times D$ region around the point to track.
- Use SSD, SAD, or NCC



Search region

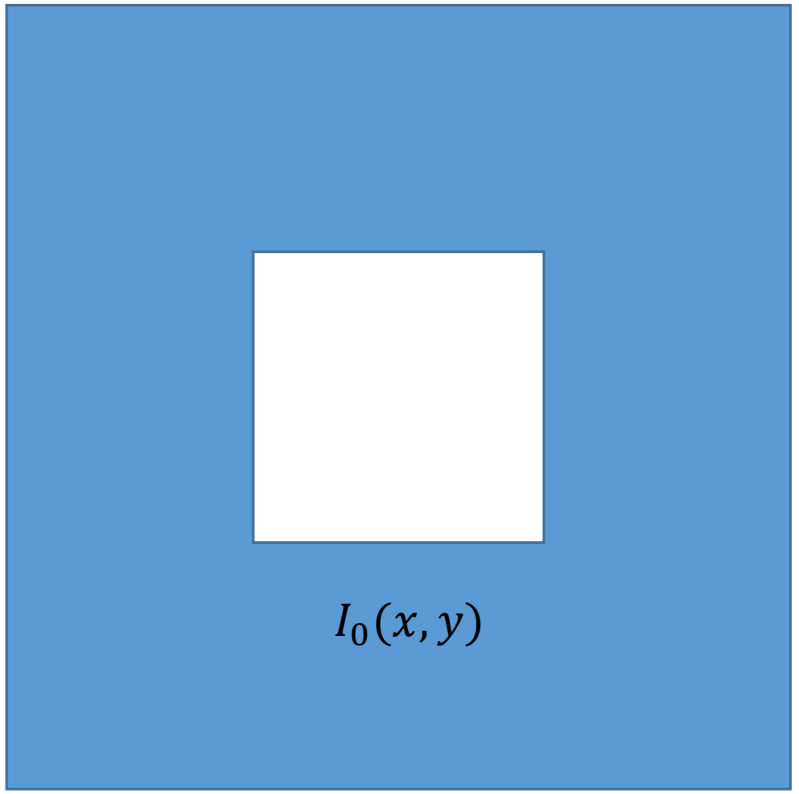
Pros and Cons of Block Matching

- Pros:
  - Works well if the motion is large
- Cons
  - Can become computationally demanding if the motion is large

- Can the “search” be implemented in a smart way if the motion is “small”?
  - Yes, use Differential methods

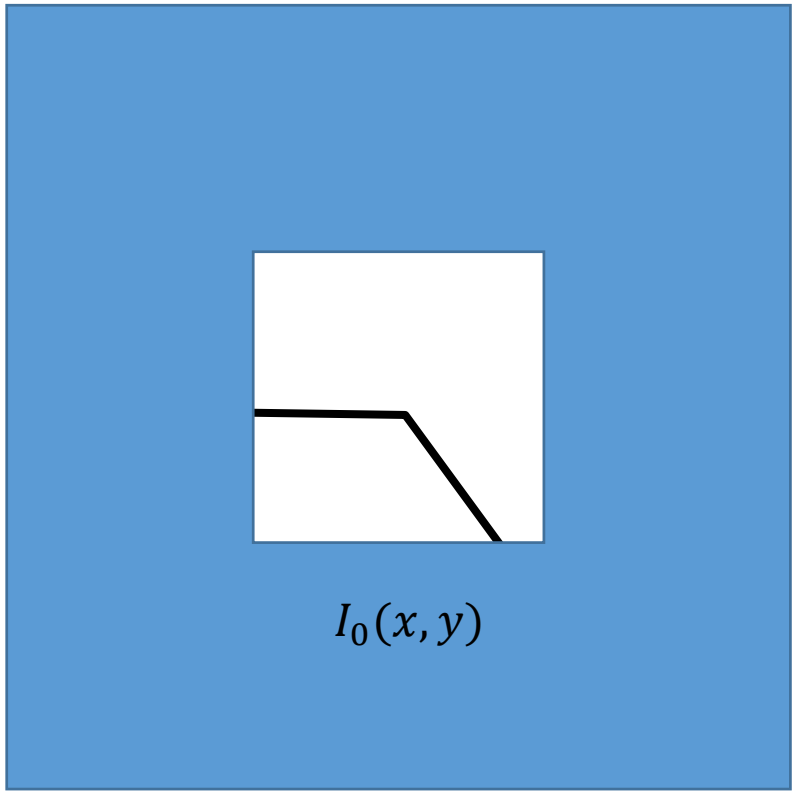
Point Tracking with Differential Methods

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



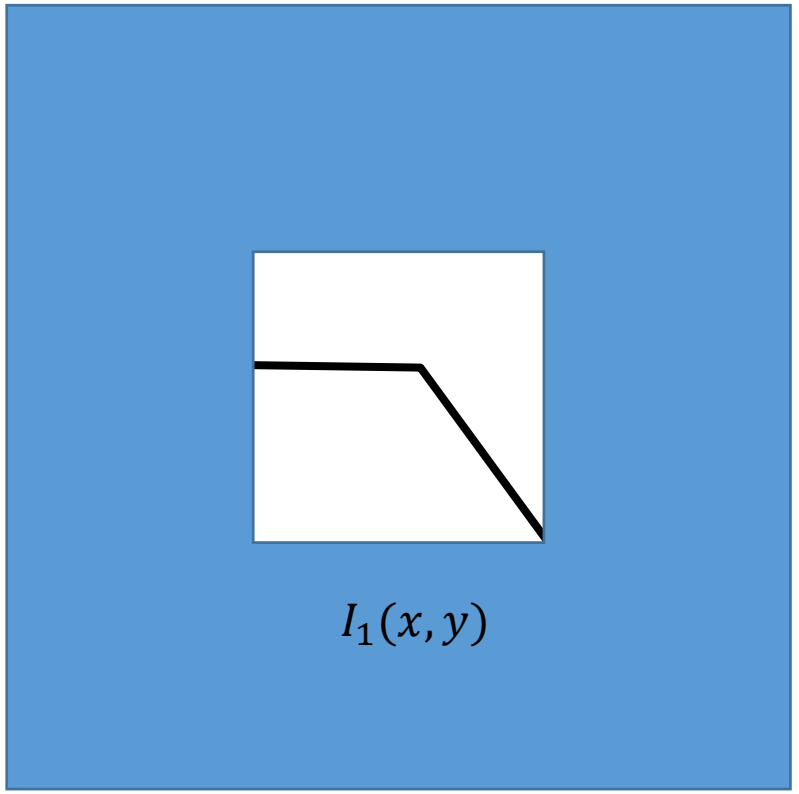
Point Tracking with Differential Methods

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



Point Tracking with Differential Methods

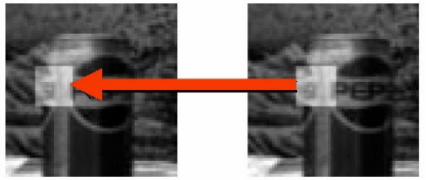
Looks at the local brightness changes at the **same** location. **No patch shift** is performed!



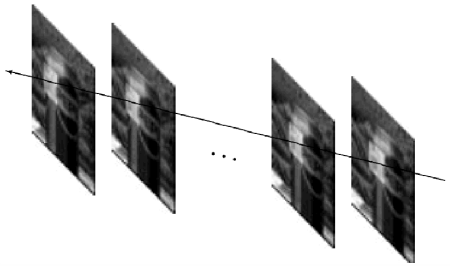
Point Tracking with Differential Methods

Assumptions:

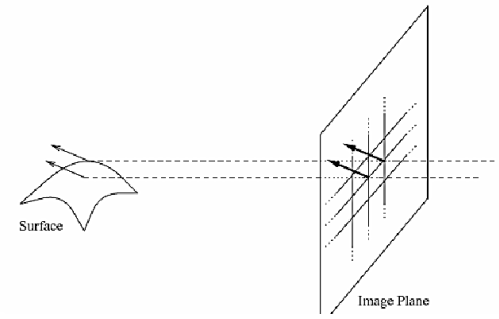
- **Brightness constancy**
  - The intensity of the pixels around the point to track does not change much between the two frames



- **Temporal consistency**
  - The motion displacement is small (1-2 pixels); however, this can be addressed using multi-scale implementations (see later)

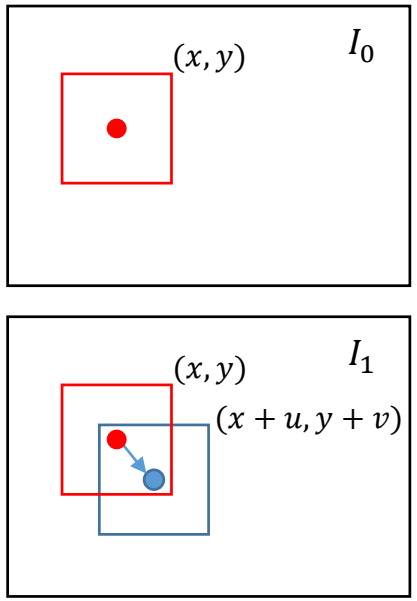


- **Spatial coherency**
  - Neighboring pixels undergo similar motion (i.e., they all lay on the same 3D surface, i.e., no depth discontinuity)



The Kanade-Lucas-Tomasi (KLT) tracker

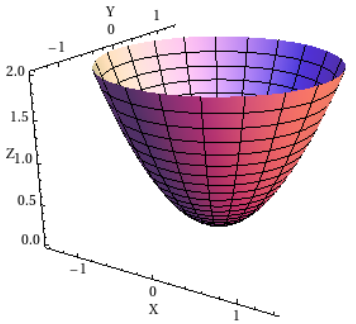
Consider the reference patch centered at (x, y) in image I_0 and the shifted patch centered at $(x + u, y + v)$ in image I_1. The patch has size Ω. We want to find the motion vector (u, v) that minimizes the Sum of Squared Differences (SSD):



$$SSD(u, v) = \sum_{x,y \in \Omega} (I_0(x, y) - I_1(x + u, y + v))^2$$

$$\cong \sum (I_0(x, y) - I_1(x, y) - I_x u - I_y v)^2$$

$$\Rightarrow \; SSD(u, v) = \sum (\Delta I - I_x u - I_y v)^2$$



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

Lucas, Kanade, *An iterative image registration technique with an application to stereo vision.* Proceedings of Imaging Understanding Workshop, 1981. PDF.
Tomasi, Kanade, *Detection and Tracking of Point Features*, Carnegie Mellon University Technical Report CMU-CS-91-132, 1991. PDF.

The Kanade-Lucas-Tomasi (KLT) tracker

$$SSD(u,v) = \sum (\Delta I - I_x u - I_y v)^2$$

To minimize it, we differentiate it with respect to (u, v) and equate it to zero:

$$\frac{\partial SSD}{\partial u} = 0 \, , \, \frac{\partial SSD}{\partial v} = 0$$

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

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

The Kanade-Lucas-Tomasi (KLT) tracker

$$\sum I_x(\Delta I - I_x u - I_y v) = 0$$

$$\sum I_y(\Delta I - I_x u - I_y v) = 0$$

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

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

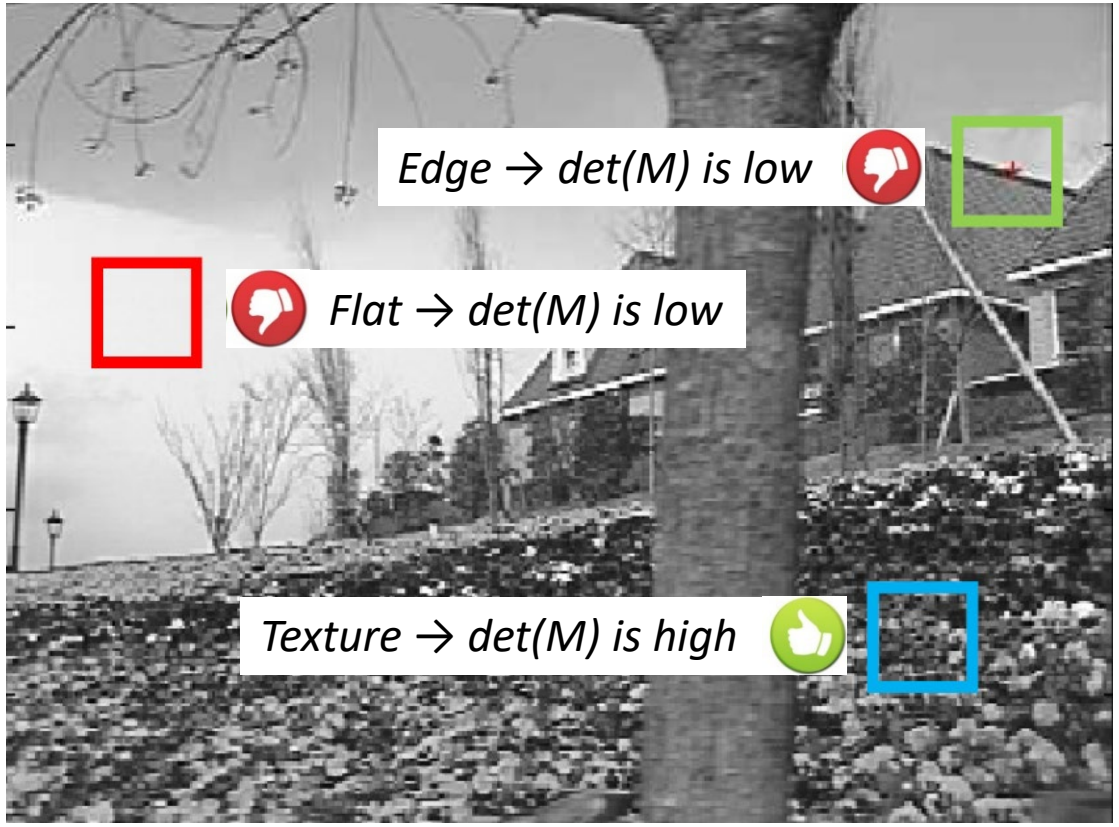
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x I_x & \sum I_x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

Haven't we seen this matrix already? Recall Harris detector!

The Kanade-Lucas-Tomasi (KLT) tracker

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) $\rightarrow$ in practice, it **should be a corner or more generally contain any textured region**!

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

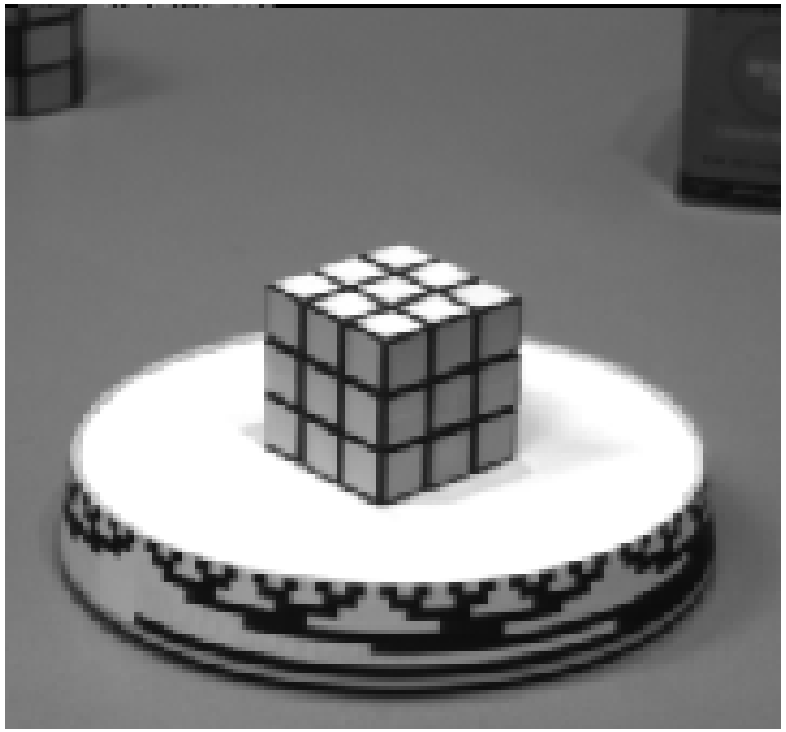


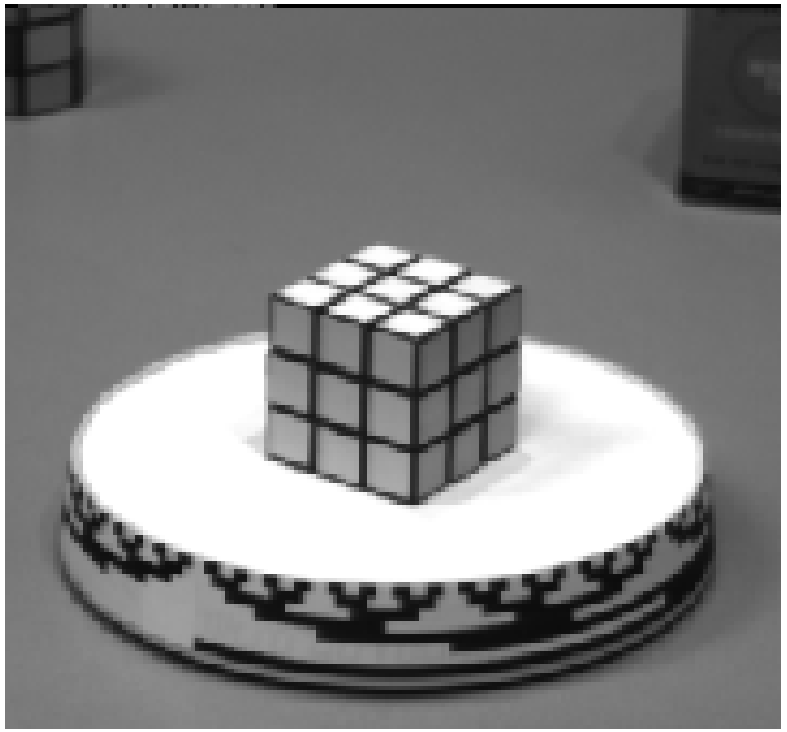
Application to Corner Tracking

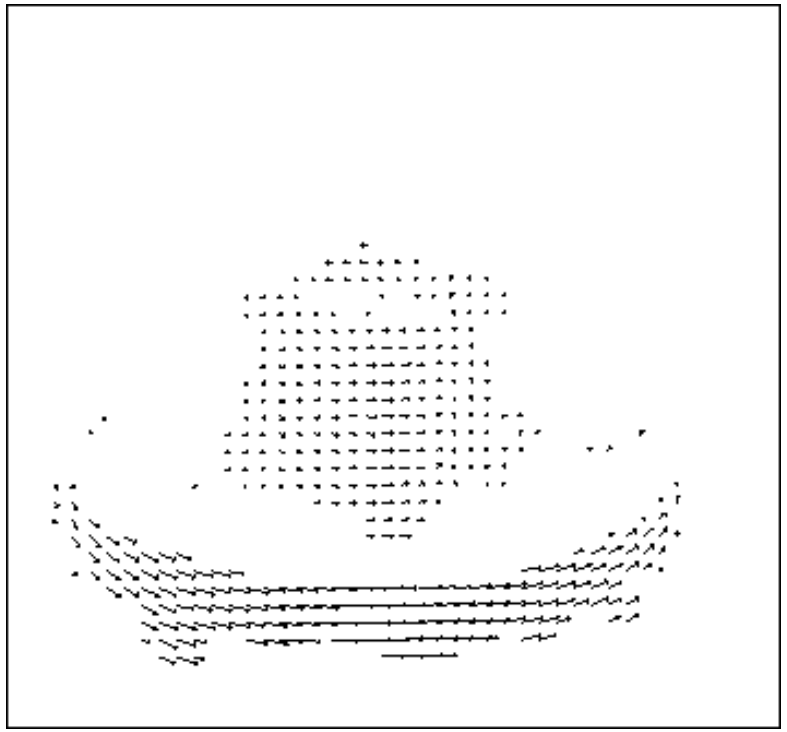
Color encodes motion direction

Application to Optical Flow

What if you track every single pixel in the image?







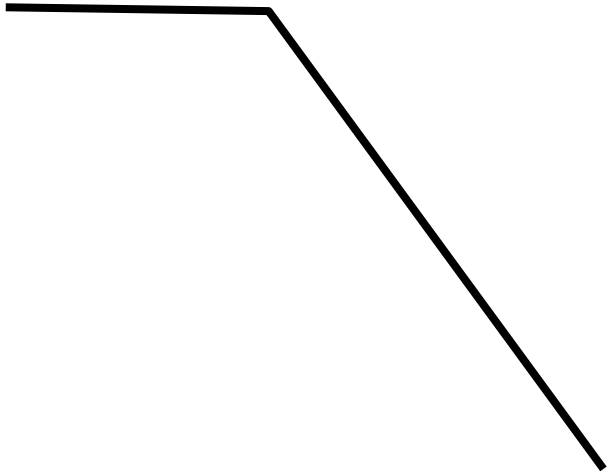
Application to Optical Flow

Application to Optical Flow

Optical Flow example

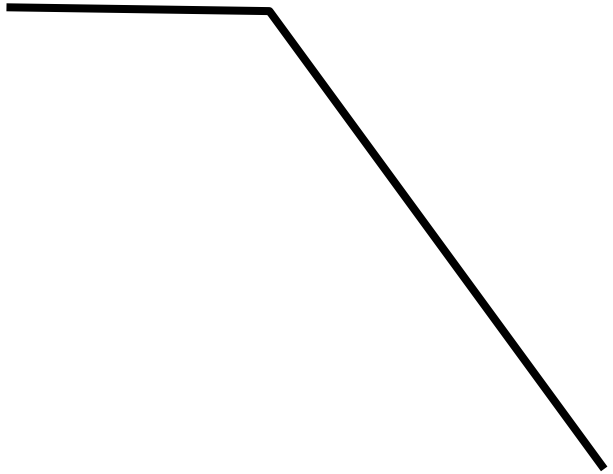
Aperture Problem

- Consider the motion of the following corner



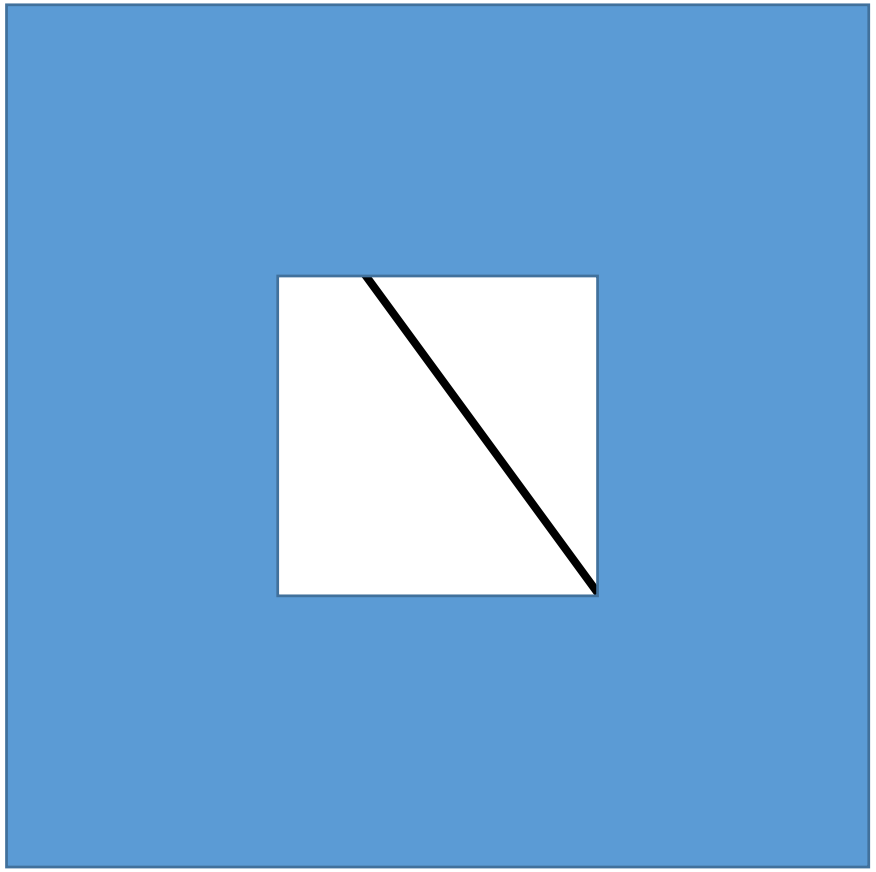
Aperture Problem

- Consider the motion of the following corner



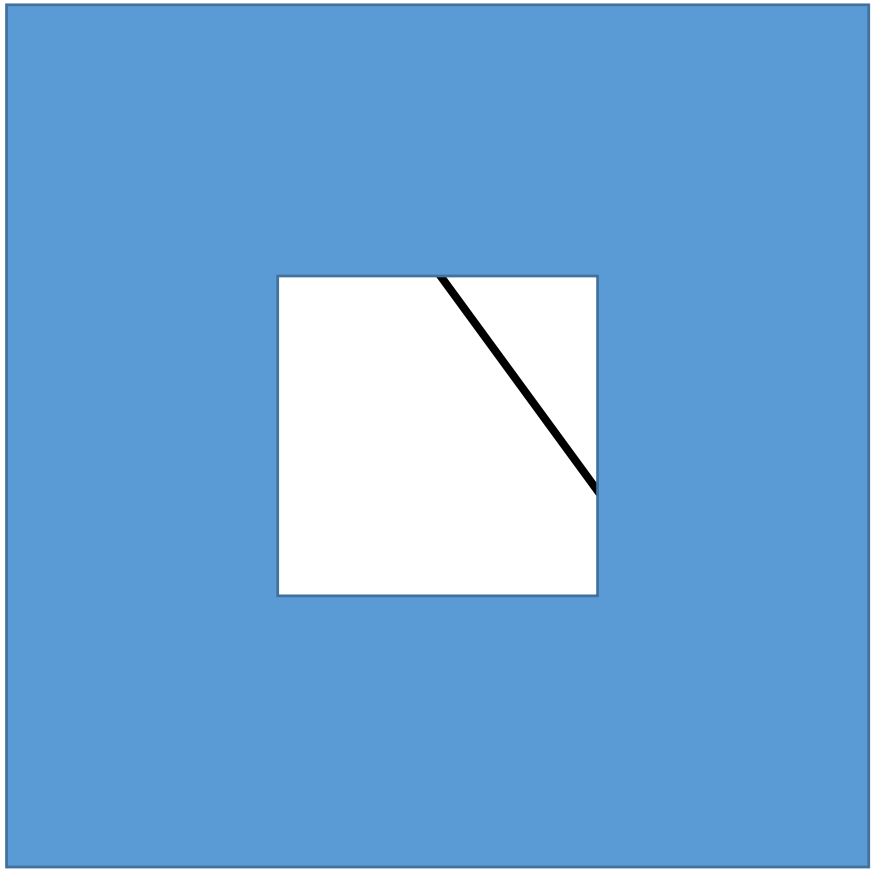
Aperture Problem

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



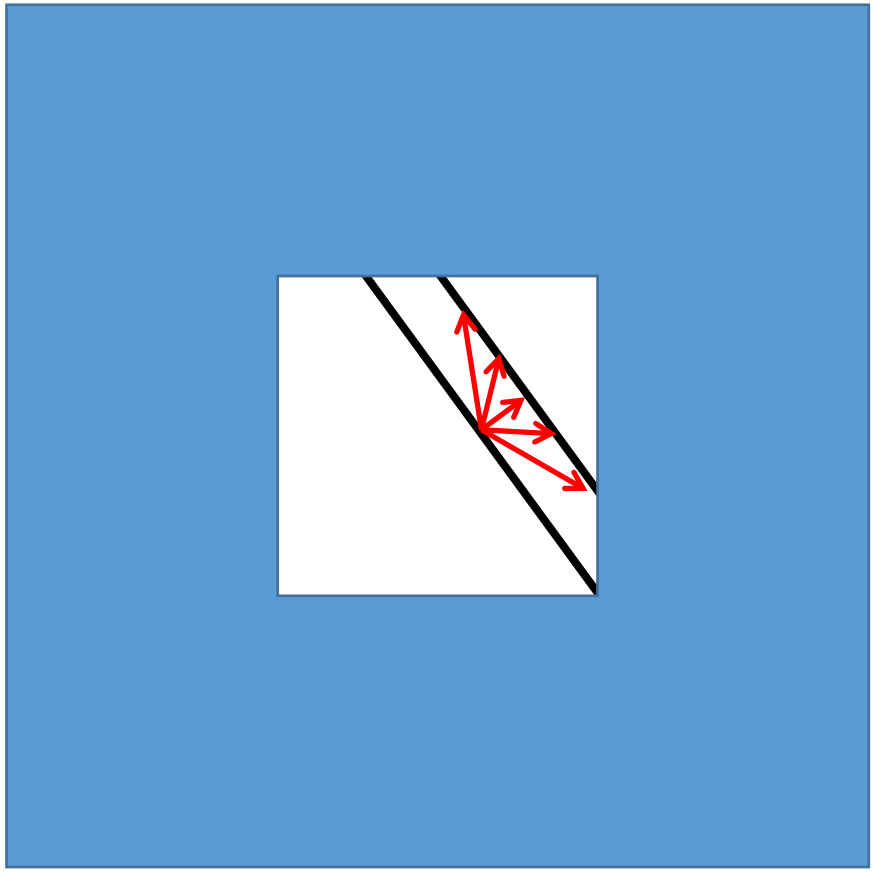
Aperture Problem

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



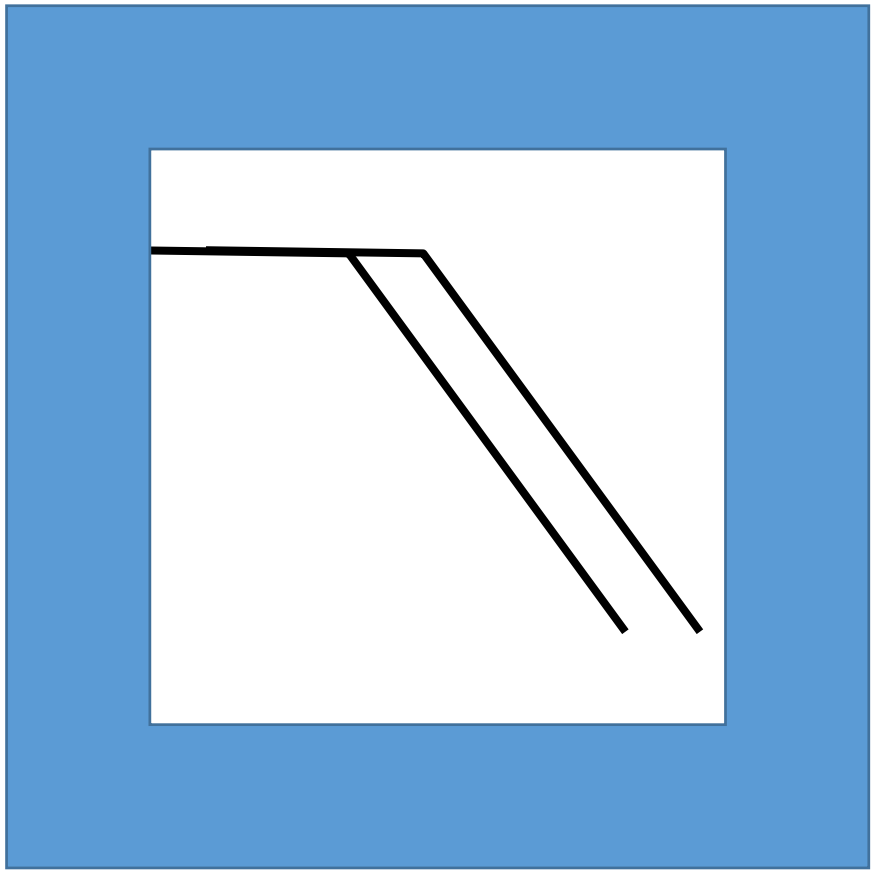
Aperture Problem

- 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!



Block-based vs Differential methods

- **Block-based methods:**
  - **Robust to large motions**
  - Can be computationally **expensive** ($D \times D$ validations need to be made for a single point to track)

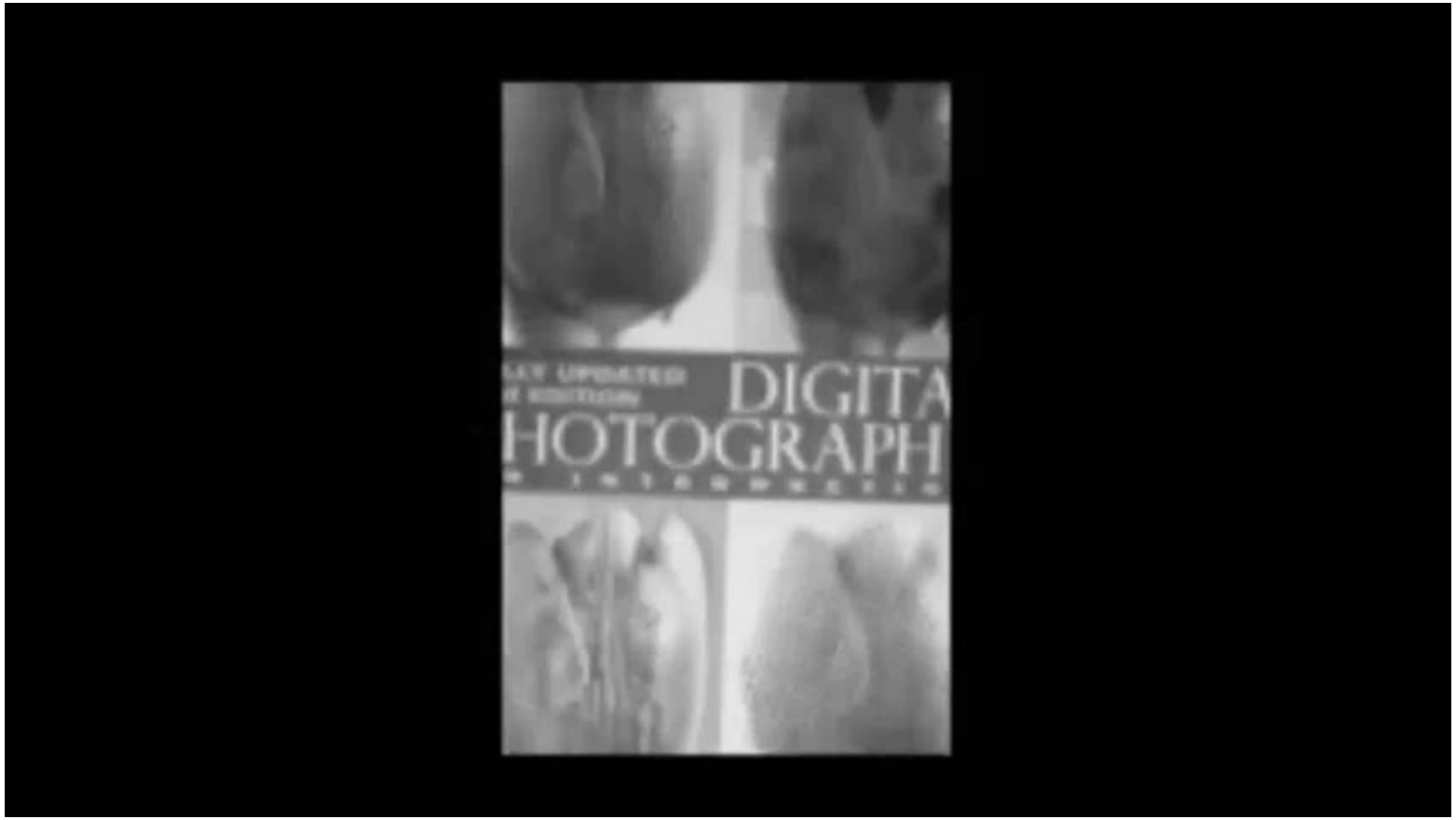
- **Differential methods:**
  - Works only for **small motions** (e.g., high frame rate). For **larger motion**, **multi-scale implementations** are used but are more expensive (see later)
  - Much **more efficient than block-based methods**. Thus, can be used to track the motion of every pixel in the 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

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

Template tracking

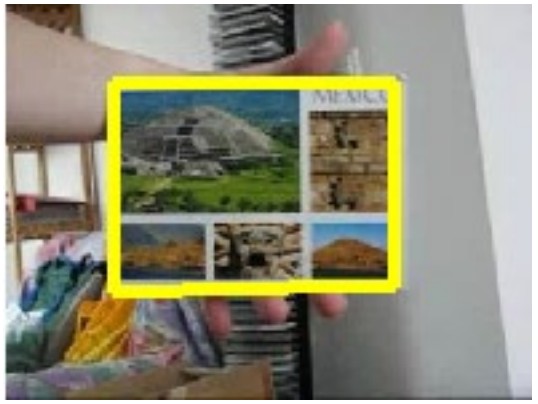
Goal: follow a template image in a video sequence by estimating the warp

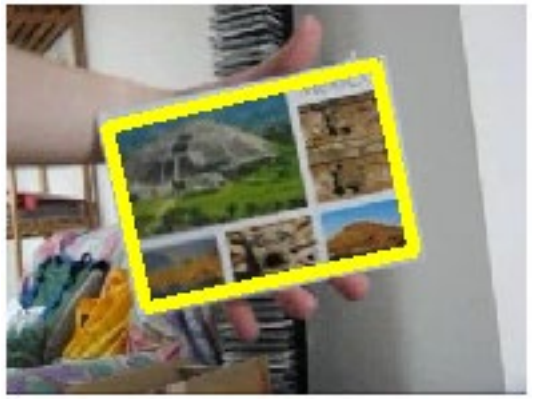


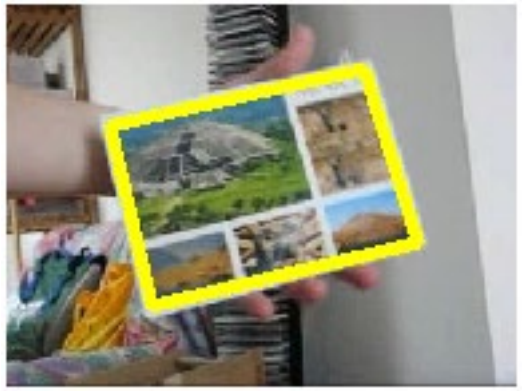
Template tracking

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

Template image



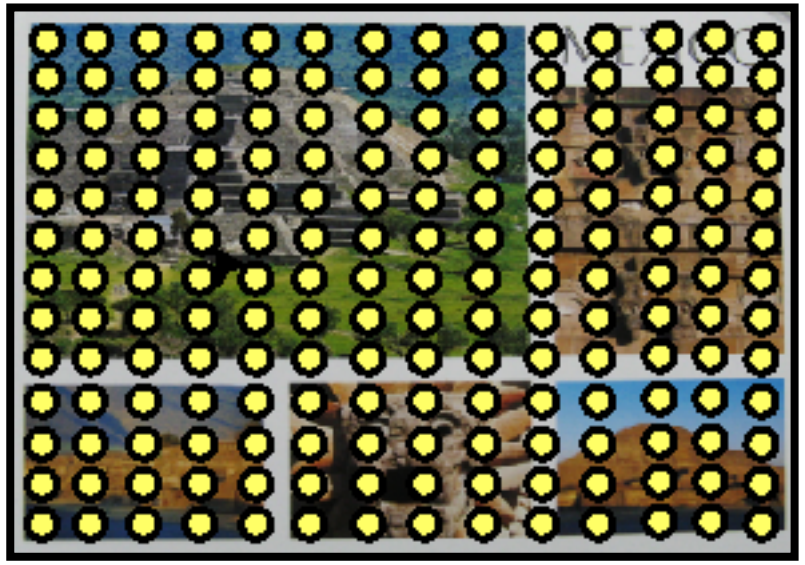




Template Warping

- 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 image

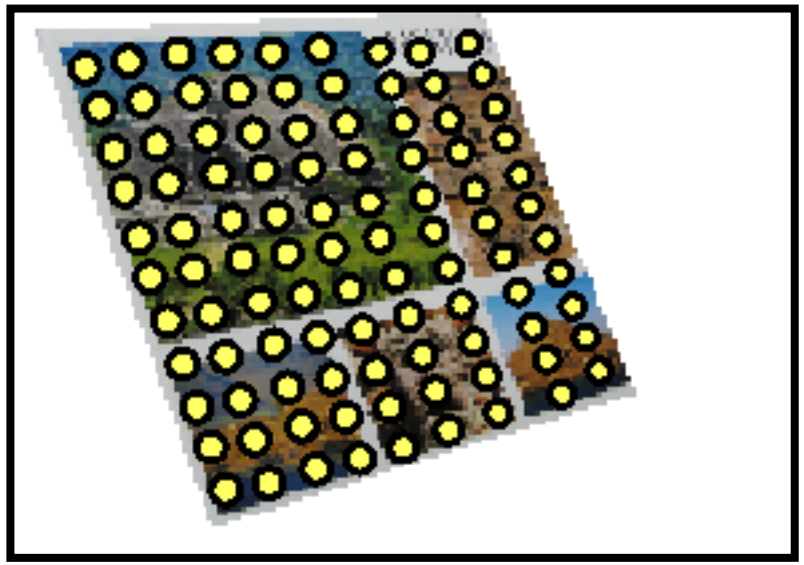


$T(\mathbf{x})$

warp

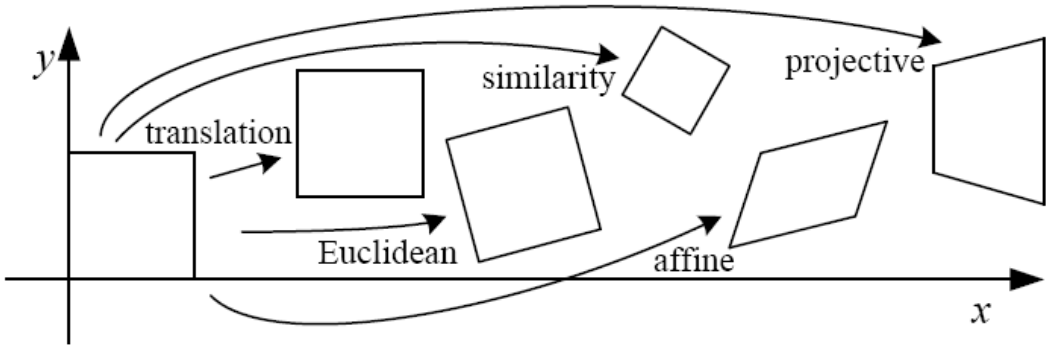
$W(\mathbf{x}, \mathbf{p})$

Current image



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

Common 2D Transformations



- Translation

$$x' = x + a_1$$
$$y' = y + a_2$$

- Euclidean

$$x' = xcos(a_3) - ysin(a_3) + a_1$$
$$y' = xsin(a_3) + ycos(a_3) + a_2$$

- Affine

$$x' = a_1 x + a_3 y + a_5$$
$$y' = a_2 x + a_4 y + a_6$$

- Projective (homography)

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1}$$

$$y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}$$

Common 2D Transformations in Matrix form

We denote the transformation $W(\mathbf{x}, \mathbf{p})$ 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

$$W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} xcos(a_3) - ysin(a_3) + a_1 \\ xsin(a_3) + ycos(a_3) + a_2 \end{bmatrix} = \begin{bmatrix} \cos(a_3) & -\sin(a_3) & a_1 \\ \sin(a_3) & \cos(a_3) & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 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 (homography)

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

Common 2D Transformations in Matrix form

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times3}$	8	straight lines	

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

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

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

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

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

Derivative and gradient

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

Jacobian

- $F(x_1, x_2, \ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_m(x_1, x_2, \ldots, x_n) \end{bmatrix}$ is a vector-valued function

- The derivative in this case is called Jacobian $\frac{\partial F}{\partial \mathbf{x}}$:

$$\frac{\partial F}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \ldots, \frac{\partial f_1}{\partial x_n} \\ \vdots \\ \frac{\partial f_m}{\partial x_1}, \ldots, \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Carl Gustav Jacob (1804-1851)

Displacement-model Jacobians ∇W_p

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

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

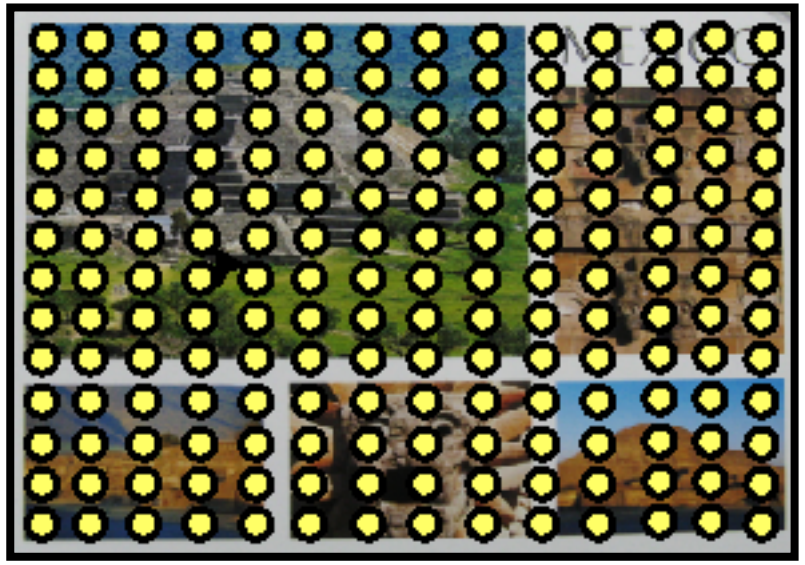
- Euclidean: $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} xcos(a_3) - ysin(a_3) + a_1 \\ xsin(a_3) + ycos(a_3) + a_2 \end{bmatrix}$ $\quad \frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 & -xsin(a_3) - ycos(a_3) \\ 0 & 1 & xcos(a_3) - ysin(a_3) \end{bmatrix}$

- Affine: $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} a_1 x + a_3 y + a_5 \\ a_2 x + a_4 y + a_6 \end{bmatrix}$ $\quad \frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$

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 image

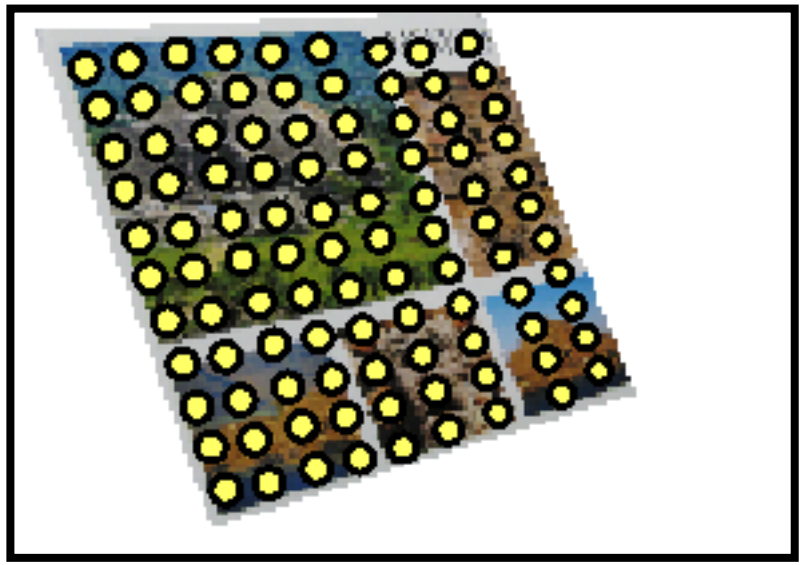


$T(\mathbf{x})$

warp

$W(\mathbf{x}, \mathbf{p})$

Current image



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

Template Tracking: Problem Formulation

- The goal of template-based tracking is to find the set of warp parameters **p** such that:

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

- This is solved by determining **p** that minimizes the Sum of Squared Differences:

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

Assumptions

- **No errors in the template image boundaries:** only the object to track appears in the template image

- **No occlusion**: the entire template is visible in the input image

- **Brightness constancy,**
- **Temporal consistency,**
- **Spatial coherency**

KLT tracker applied to template tracking

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

Lucas, Kanade, *An iterative image registration technique with an application to stereo vision.* Proceedings of Imaging Understanding Workshop, 1981. PDF.
Tomasi, Kanade, *Detection and Tracking of Point Features*, Carnegie Mellon University Technical Report CMU-CS-91-132, 1991. PDF.

Derivation of the KLT algorithm

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

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

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

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

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

$\nabla I = \left[I_x, I_y\right]$ = Image gradient evaluated at $W(\mathbf{x},\mathbf{p})$

Jacobian of the warp $W(\mathbf{x},\mathbf{p})$

Derivation of the KLT algorithm

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

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

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

- How do we minimize it?
- We differentiate SSD with respect to $\Delta\mathbf{p}$ and we equate it to zero, i.e., $\frac{\partial SSD}{\partial \Delta\mathbf{p}} = 0$

Derivation of the KLT algorithm

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

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

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

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

Derivation of the KLT algorithm

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

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

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

Second moment matrix (Hessian) of the warped image

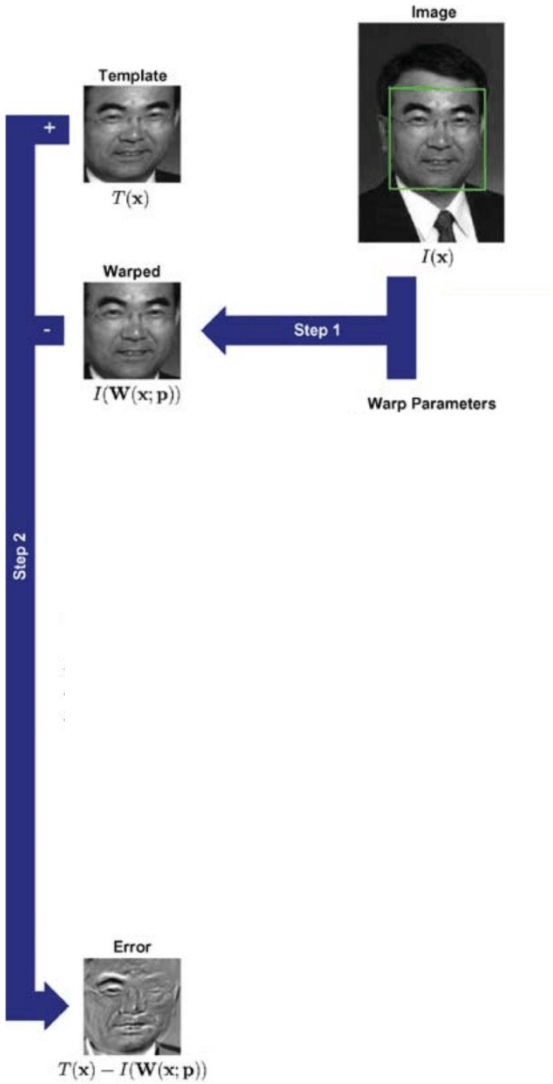
What does H look like when the warp is a pure translation?

KLT algorithm

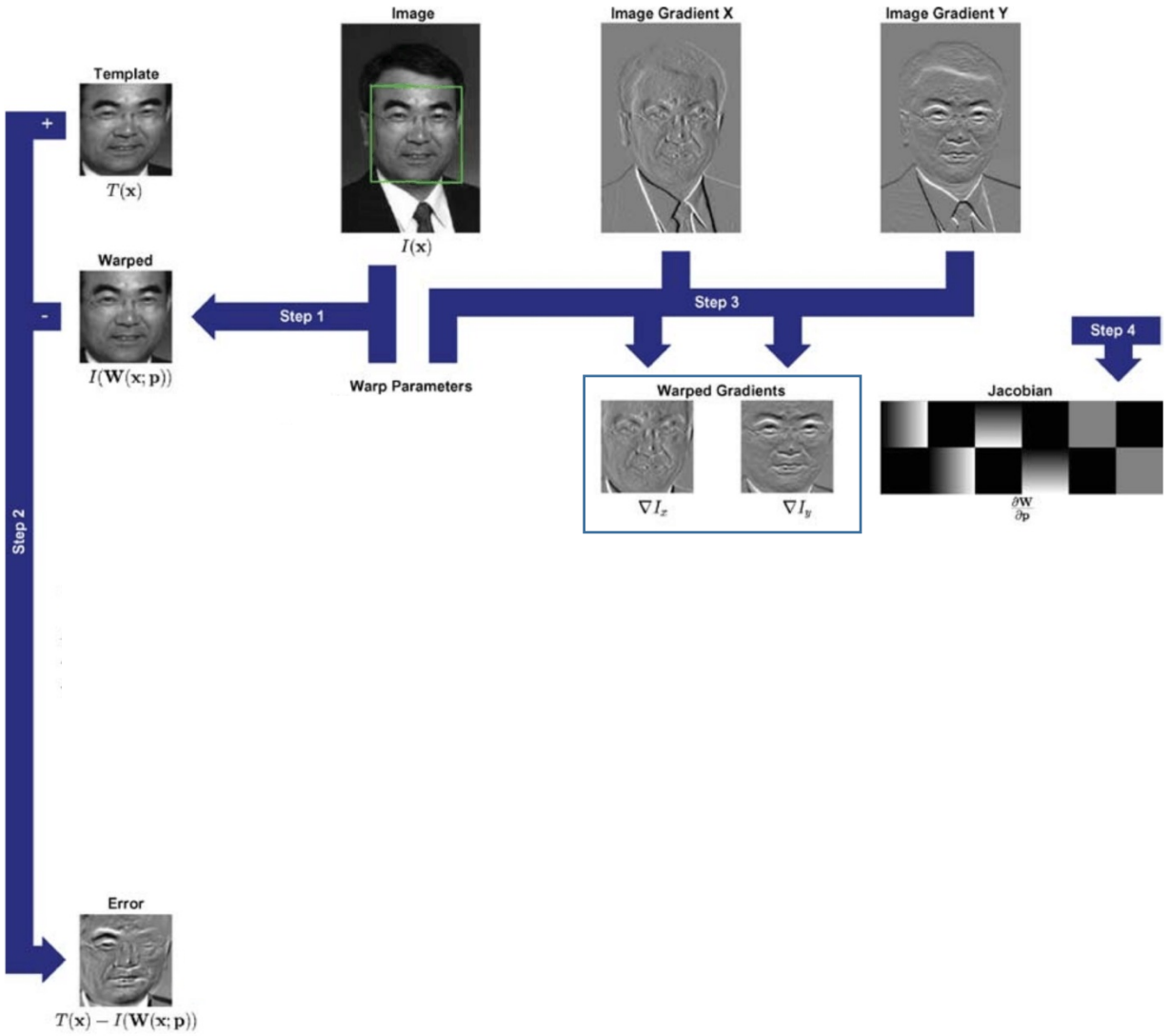
$$\Rightarrow \Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[T(\mathbf{x}) - I\big(W(\mathbf{x},\mathbf{p})\big)\right]$$

1. Warp $I(\mathbf{x})$ with $W(\mathbf{x},\mathbf{p}) \rightarrow I\big(W(\mathbf{x},\mathbf{p})\big)$
2. Compute the error: subtract $I\big(W(\mathbf{x},\mathbf{p})\big)$ from $T(\mathbf{x})$
3. Compute **warped** gradients: $\nabla I = \left[I_x, I_y\right]$, 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: $H^{-1} = \left[\sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]\right]^{-1}$
7. Multiply steepest descend with error: $\sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[T(\mathbf{x}) - I\big(W(\mathbf{x},\mathbf{p})\big)\right]$
8. Compute $\Delta\mathbf{p}$
9. Update parameters: $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$
10. Repeat until $\Delta\mathbf{p} < \boldsymbol{\varepsilon}$

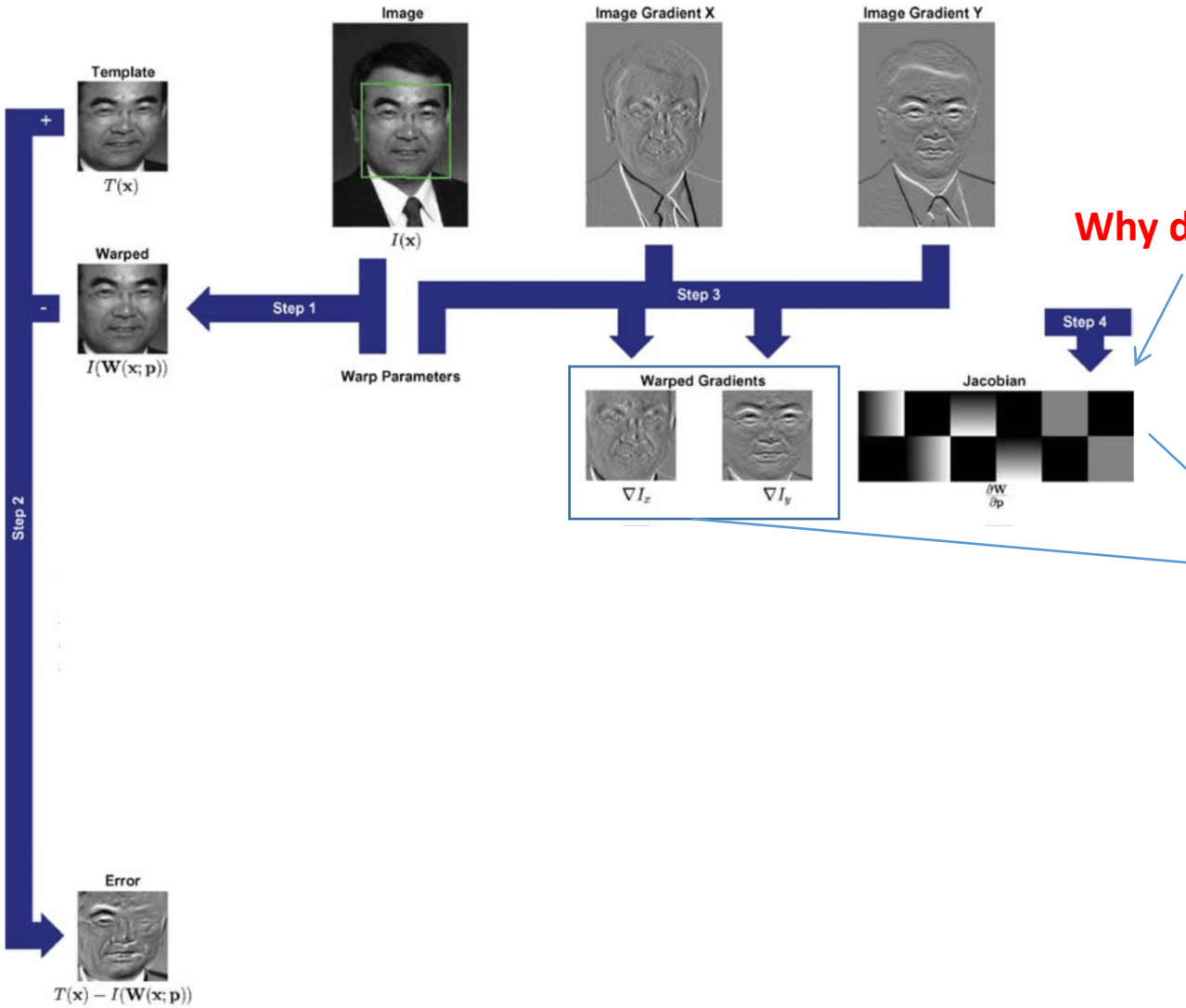
KLT algorithm: computing $\Delta\mathbf{p} = H^{-1}\sum_{\mathbf{x}\in\mathbf{T}}\left[\nabla I\frac{\partial W}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[T(\mathbf{x}) - I\left(W(\mathbf{x},\mathbf{p})\right)\right]$



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



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



Why does it look like that?

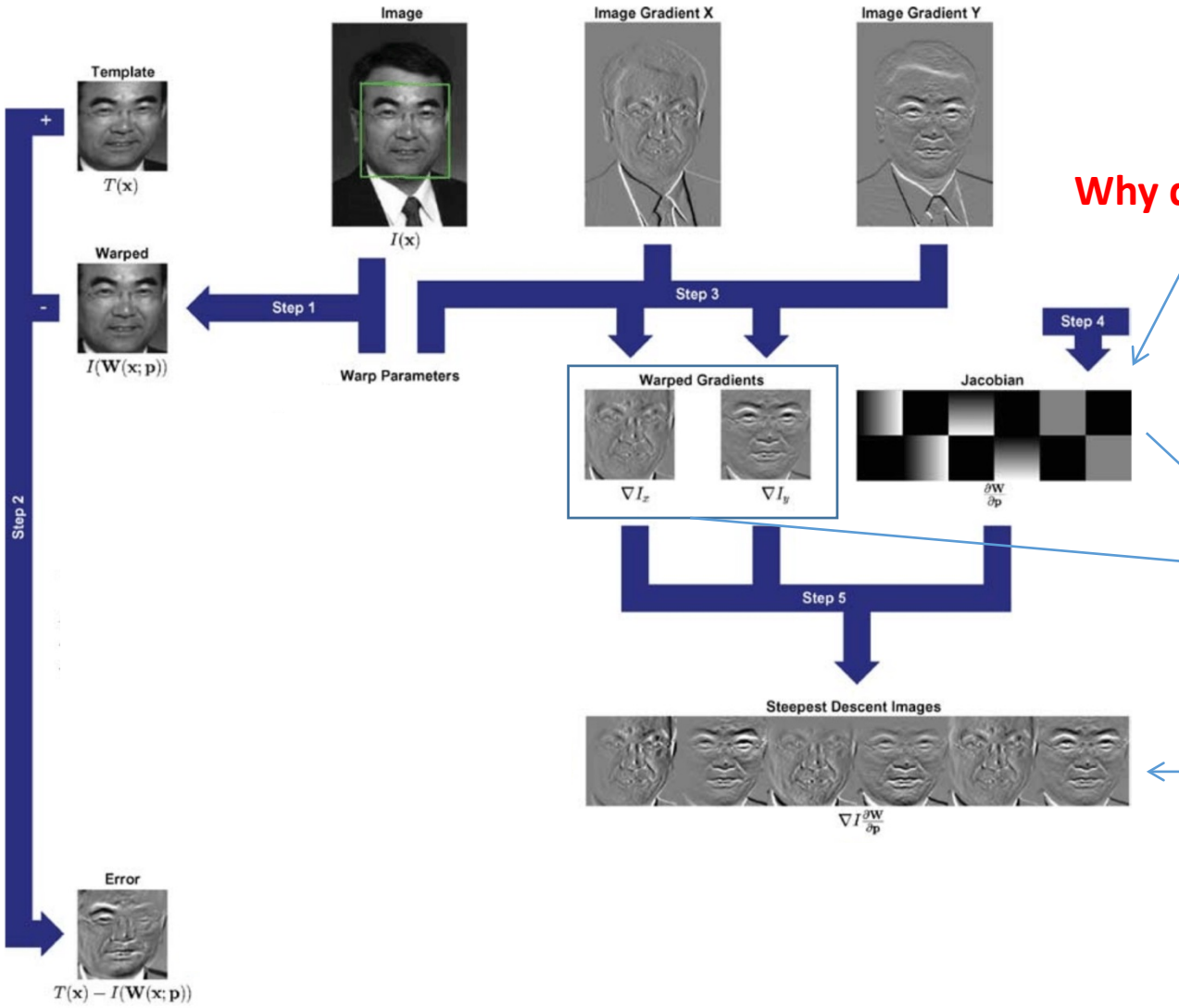
$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

What is the size?

$\boldsymbol{2n \times 6n}$

$\boldsymbol{n \times 2n}$

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



Why does it look like that? (slide 48)

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

What is the size?

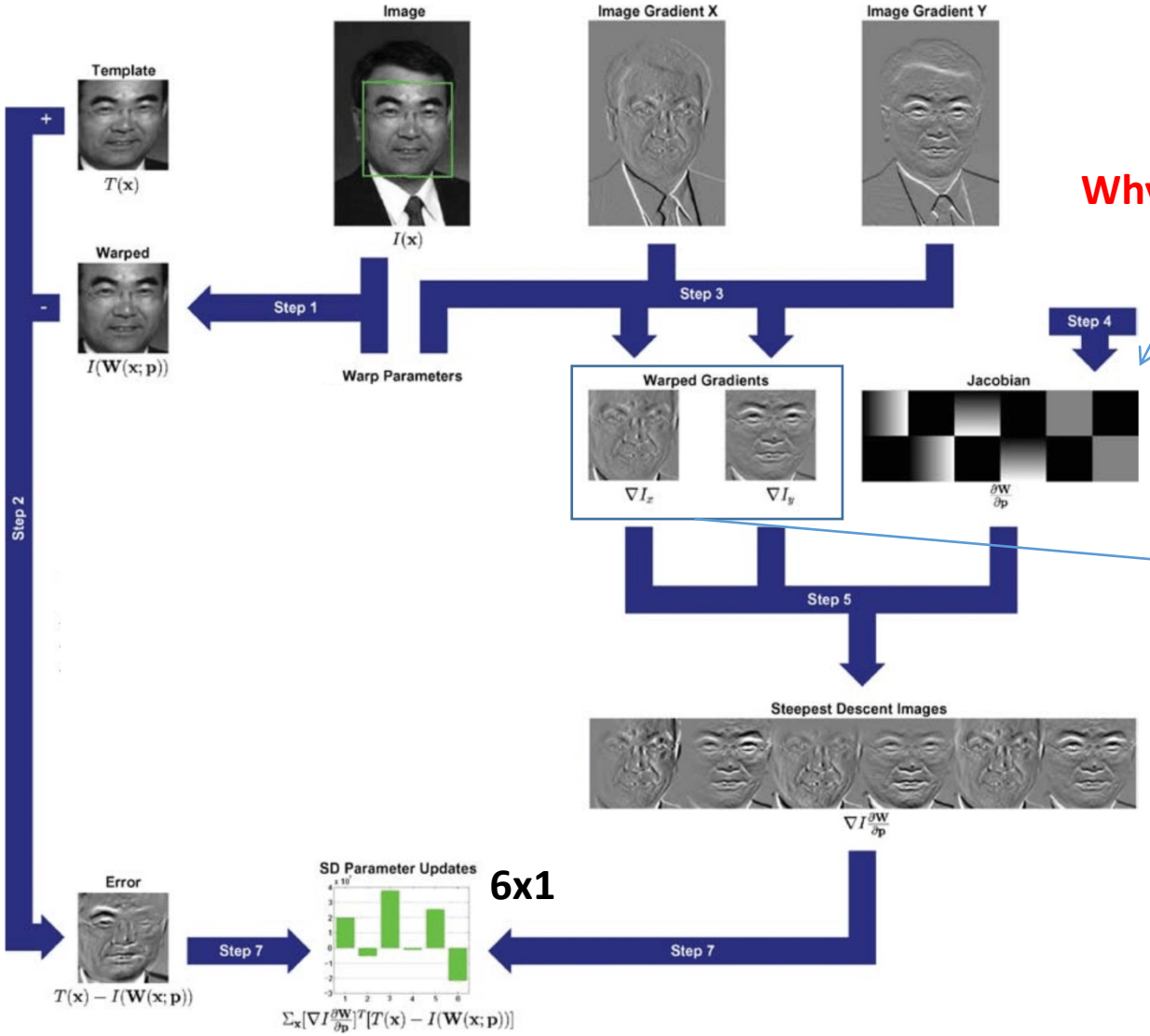
$2n \times 6n$

$n \times 2n$

What's its size?

$n \times 6n$

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



Why does it look like that? (slide 48)

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

What is the size?

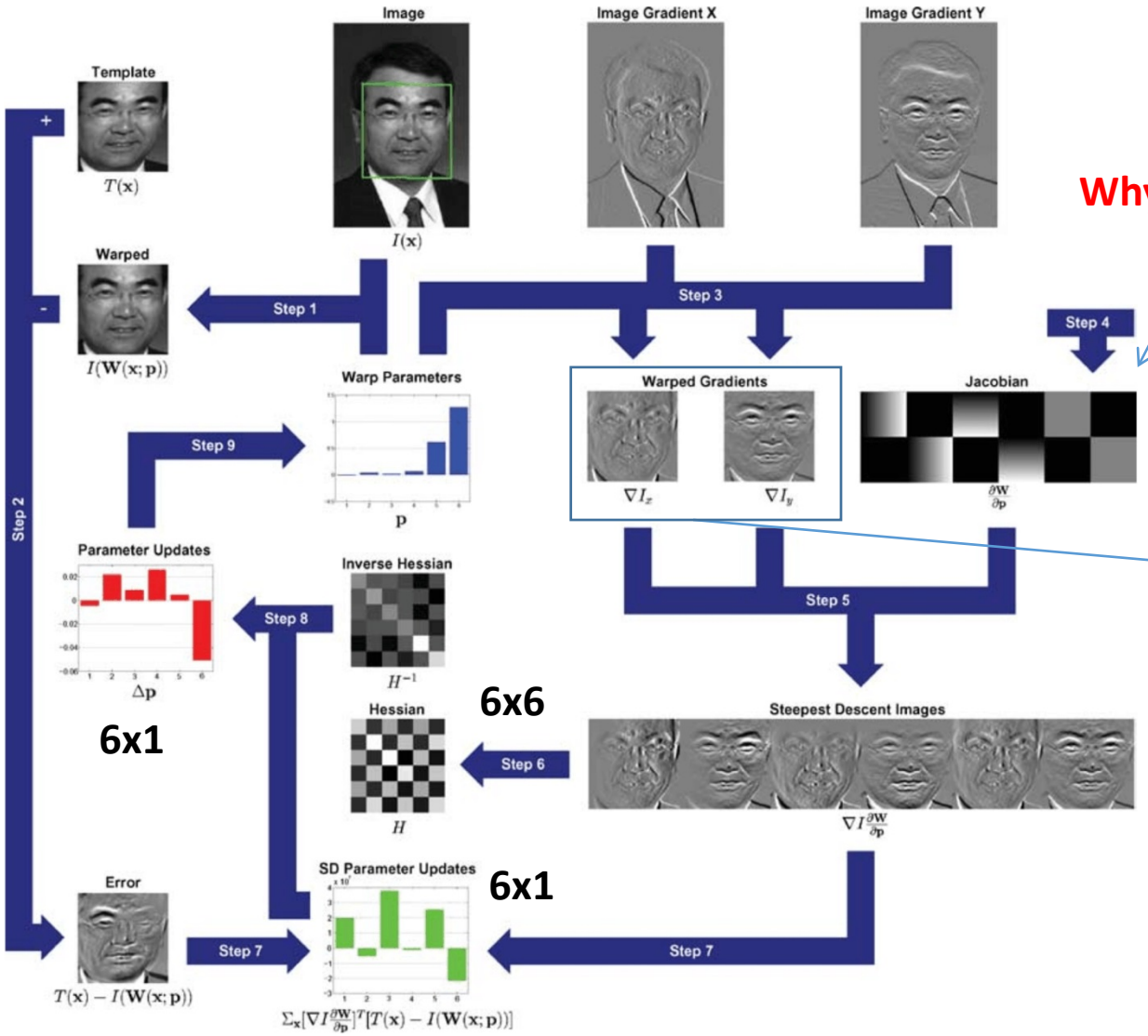
$2n \times 6n$

$n \times 2n$

What's its size?

$n \times 6n$

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



Why does it look like that?
(slide 48)

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

What is the size?

$2n \times 6n$

$n \times 2n$

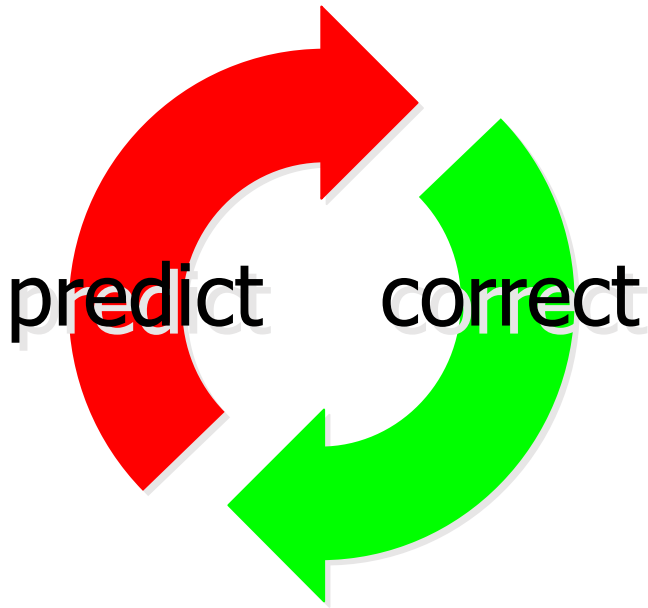
What's its size?

$n \times 6n$

KLT algorithm: Discussion

Lucas-Kanade follows a **predict-correct cycle**

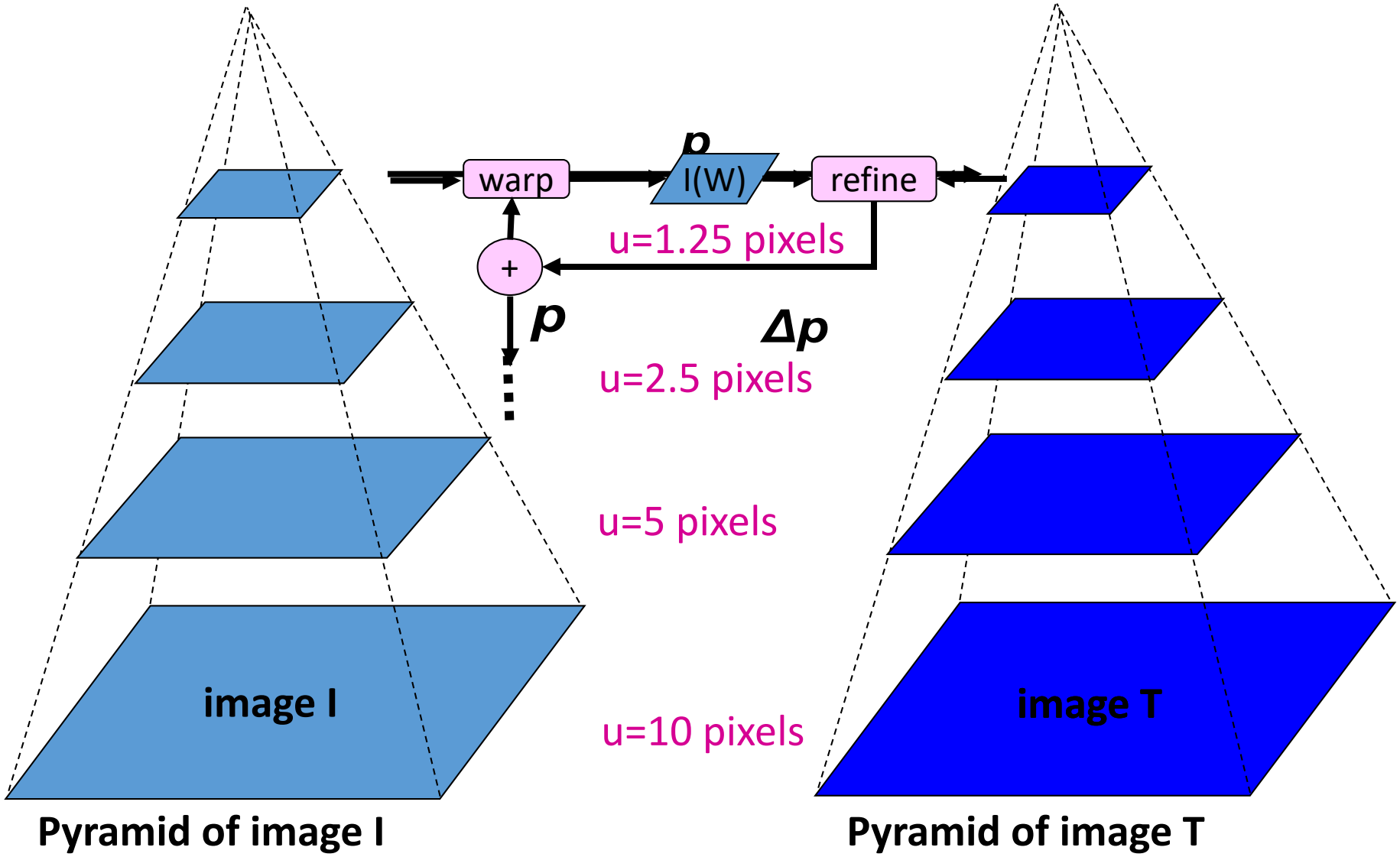
- A **prediction** $I\big(W(\mathbf{x},\mathbf{p})\big)$ of the warped image is computed from an initial estimate
- The **correction** parameter $\Delta\mathbf{p}$ is computed as a function of the error $T(\mathbf{x}) - I\big(W(\mathbf{x},\mathbf{p})\big)$ between the prediction and the template
- The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



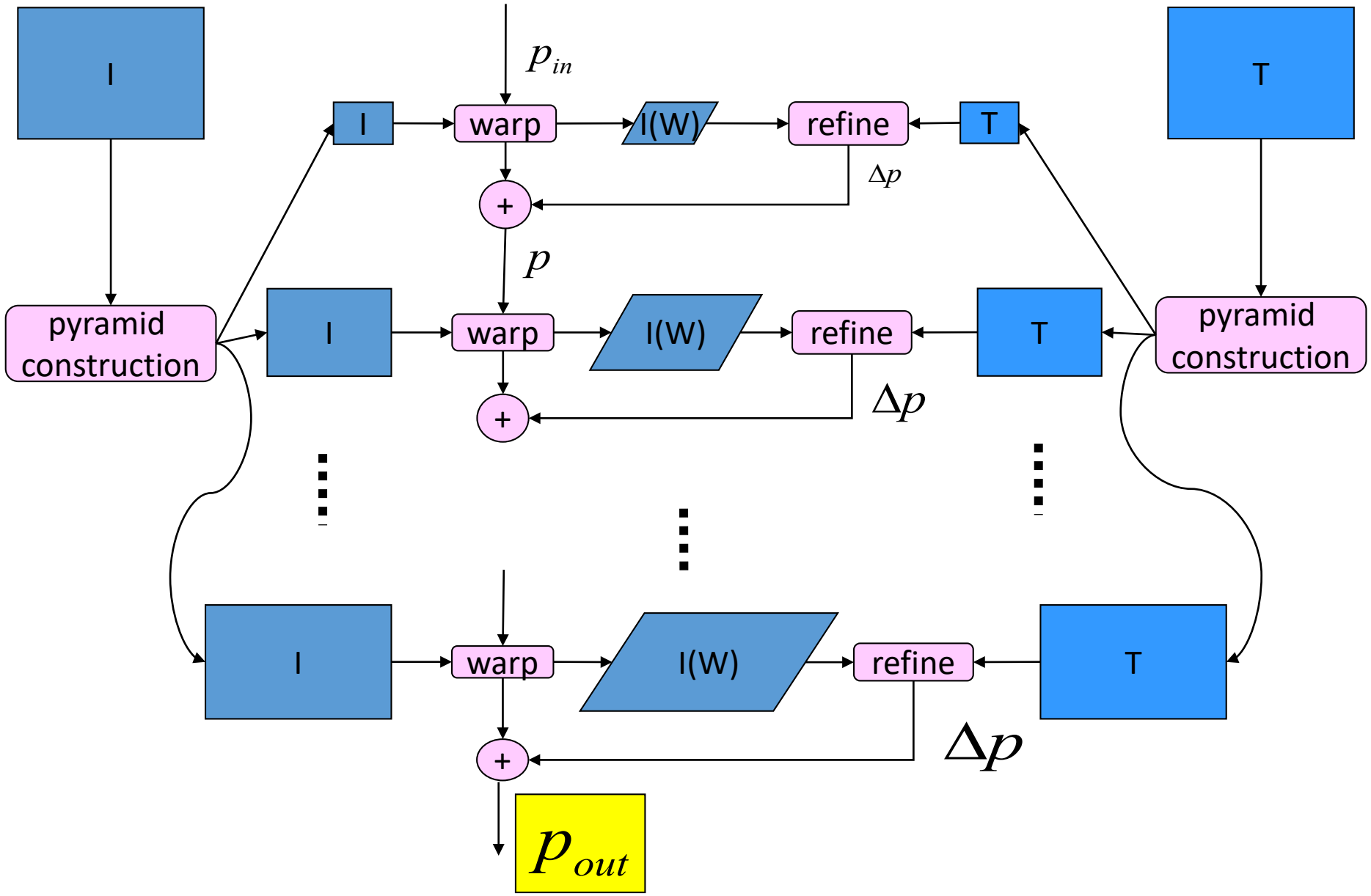
KLT algorithm: Discussion

- How to get the initial estimate **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

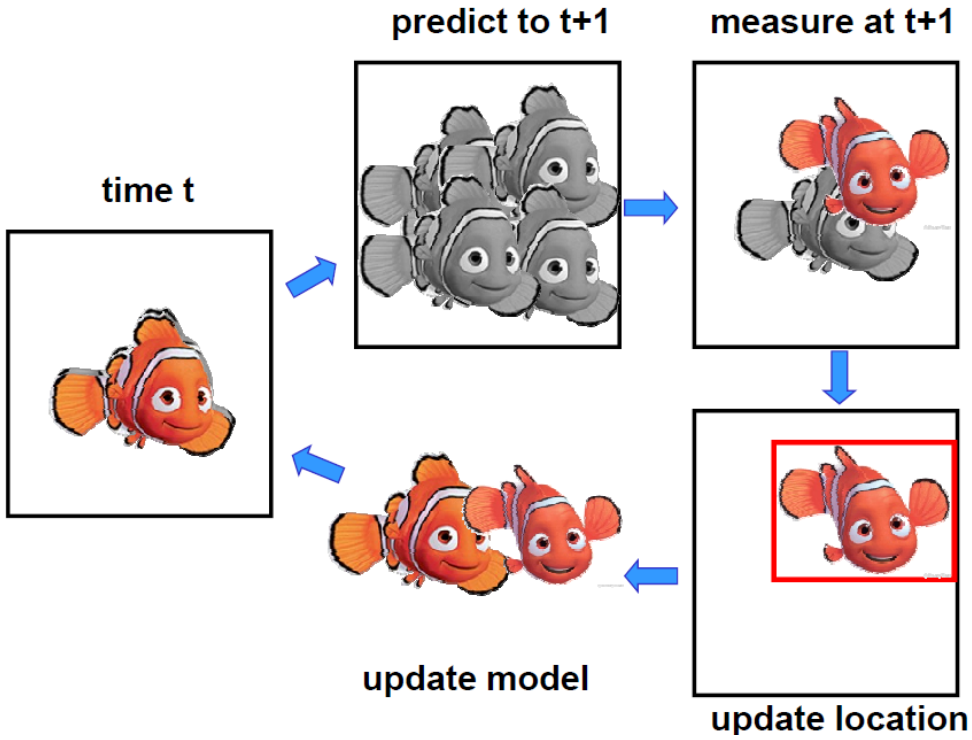


Generalization of KLT

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

Generalization of KLT

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

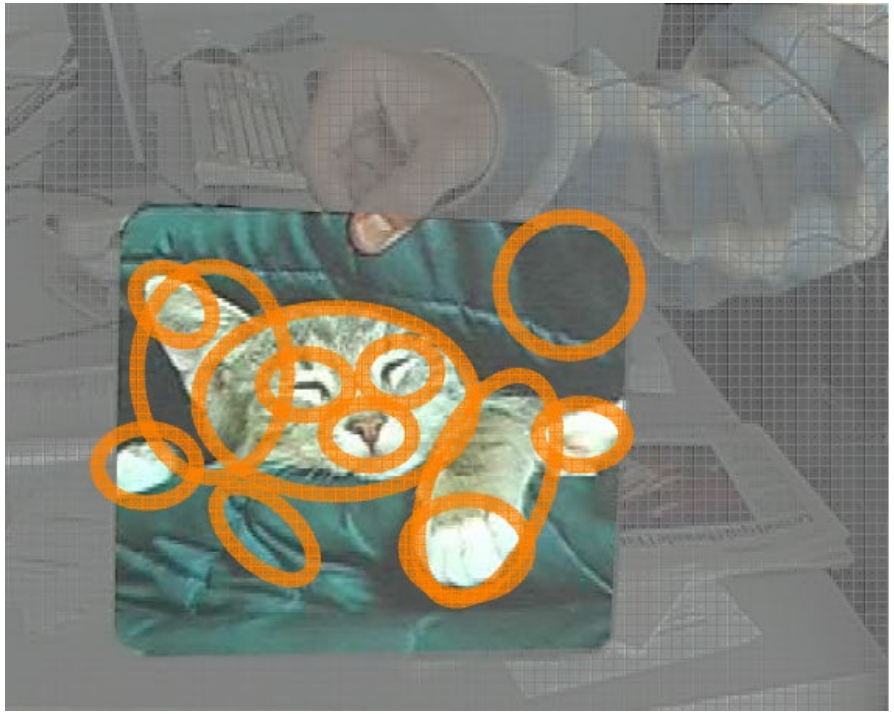


Outline

- 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

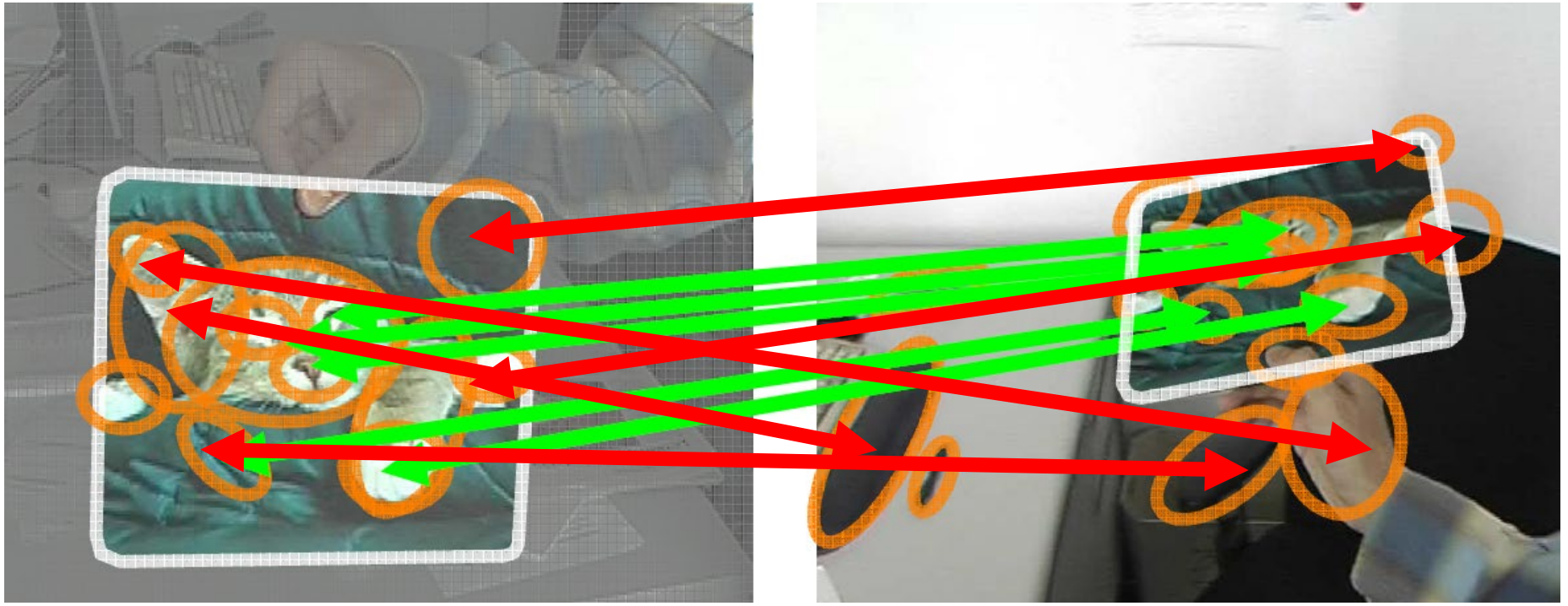


Template image with the object to detect

Current test image

Tracking by detection of local image features

- Step 1: Keypoint detection and matching
  - invariant to scale, rotation, or perspective

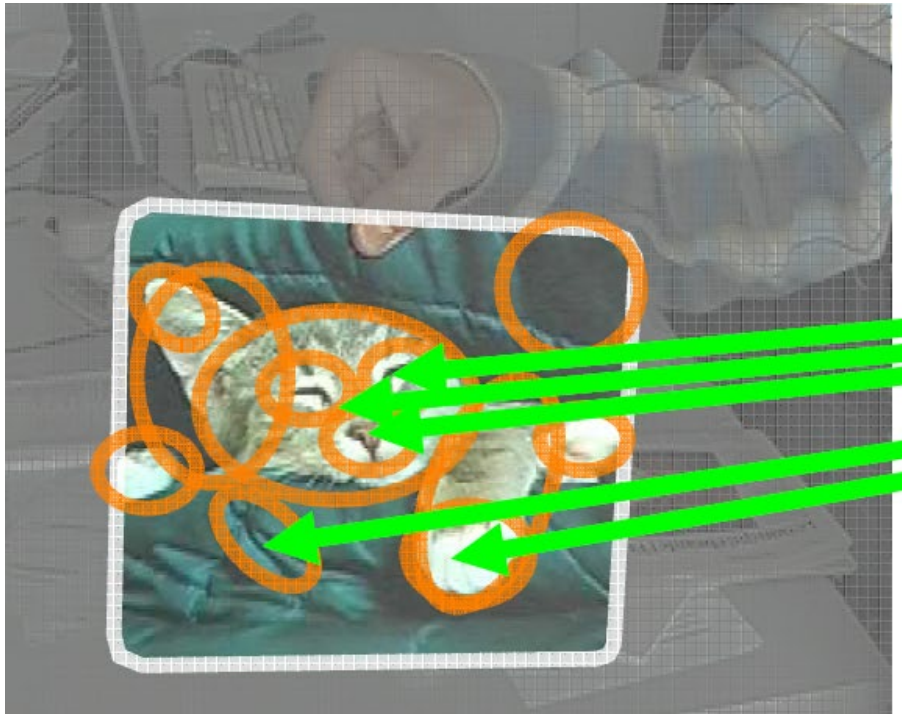


Template image with the object to detect

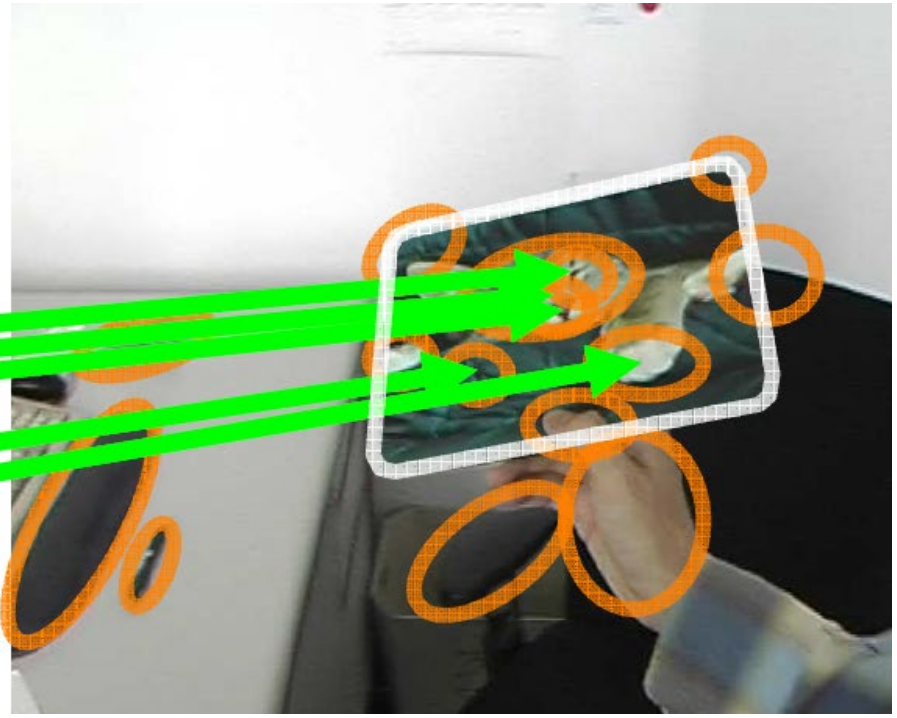
Current test image

Tracking by detection of local image features

- Step 1: Keypoint detection and matching
  - invariant to scale, rotation, or perspective
- Step 2: Geometric verification (RANSAC) (e.g., 4-point RANSAC for planar objects, or 5 or 8-point RANSAC for 3D objects)

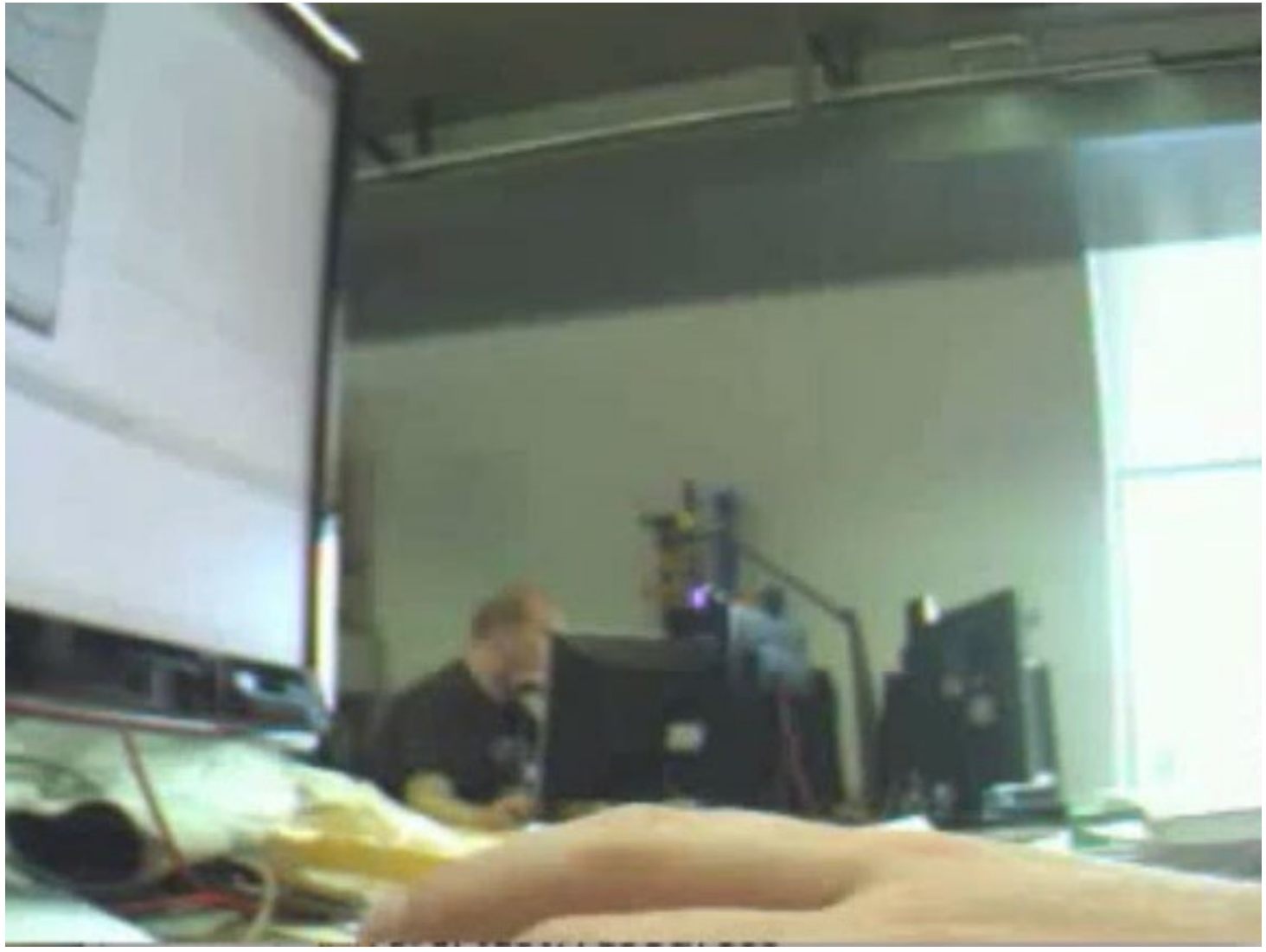


Template image with the object to detect



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?

Readings

- Chapter 8 of Szeliski's book, 1st edition

Understanding Check

Are you able to answer the following questions?

- Are you able to illustrate tracking with block matching?
- Are you able to explain the underlying assumptions behind differential methods, derive their mathematical expression and the meaning of the M matrix?
- When is this matrix invertible and when not?
- What is the aperture problem and how can we overcome it?
- What is optical flow?
- Can you list pros and cons of block-based vs. differential methods for tracking?
- Are you able to describe the working principle of KLT?
- What functional does KLT minimize?
- What is the Hessian matrix and for which warping function does it coincide to that used for point tracking?
- Can you list Lukas-Kanade failure cases and how to overcome them?
- How do we get the initial guess?
- Can you illustrate the coarse-to-fine Lucas-Kanade implementation?
- Can you illustrate alternative tracking procedures using point features?