Univerza v Ljubljani

ViCoS sualgnitive ystemslab

Advanced CV methods
Tracking patches

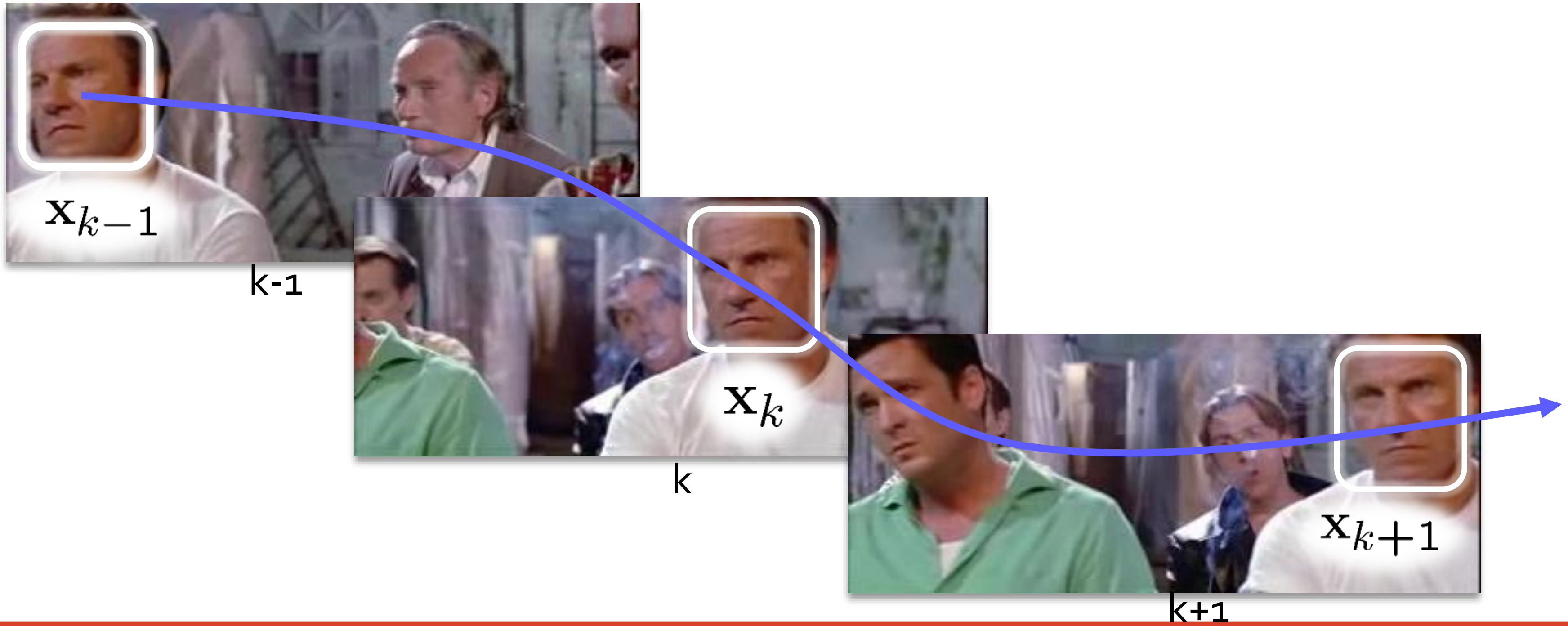
Matej Kristan

Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

Consider motion of patches of pixels

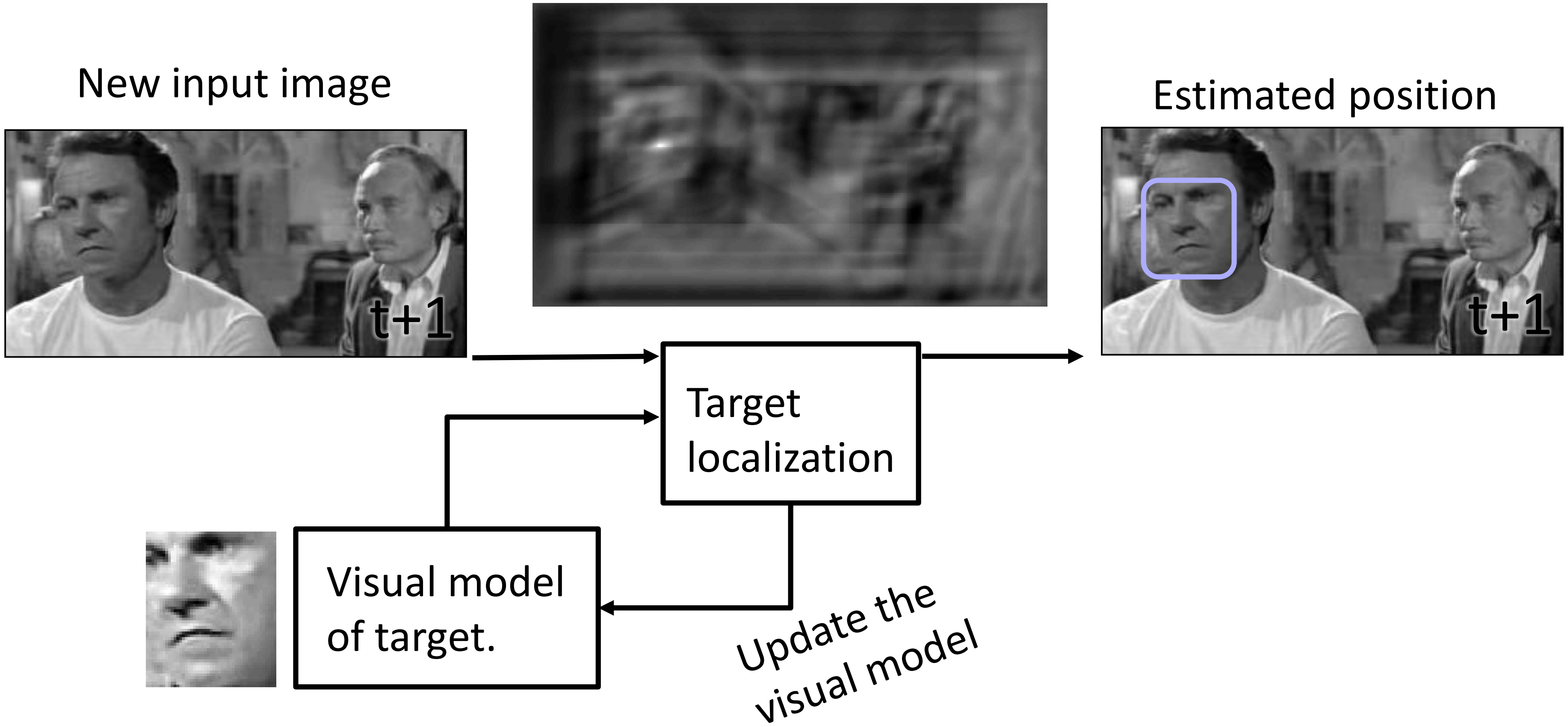
Select a region of interest in the first frame.

Assuming the object will not move by too much in consecutive frames, re-localize the object (target) in each frame.



A high-level view of tracking

- Assume some model of the target (e.g., template)
- Assume estimated position in the previous time-step



Target localization

- Correspondence problem

Extract a template $T(\boldsymbol{x})$

Previous image

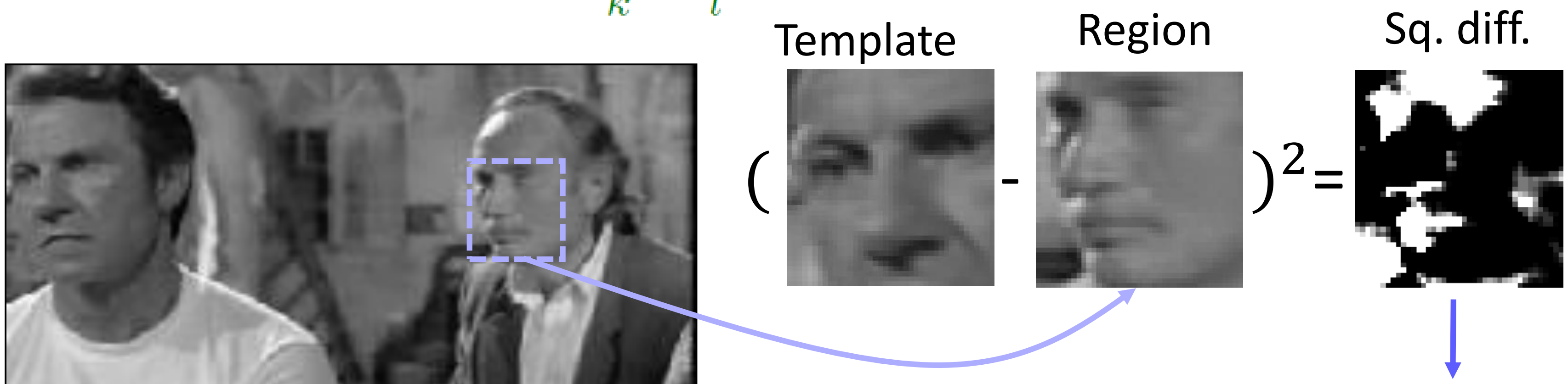
New image $I(x)$

The goal is to align a template image $T(\boldsymbol{x})$ to an input image $I(\boldsymbol{x})$.
How to measure the quality of the alignment?

Similarity measure

- Quantify the similarity between the visual model and the target region
- Straight-forward: compare pixel intensities
- “Sum of squared differences” (SSD)

$$ssd(x,y) = \sum_k \sum_l \left(T(k,l) - I(x+k, y+l)\right)^2$$

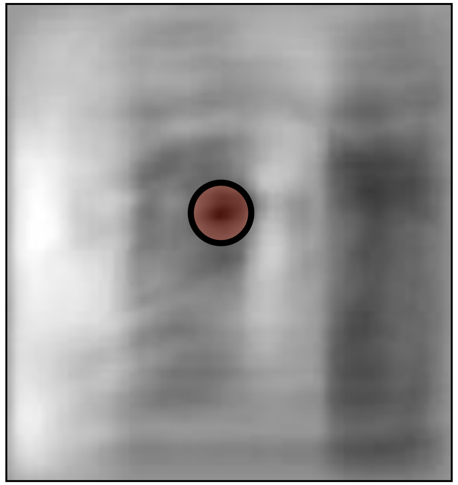


$$\sum sq.diff = 113563$$

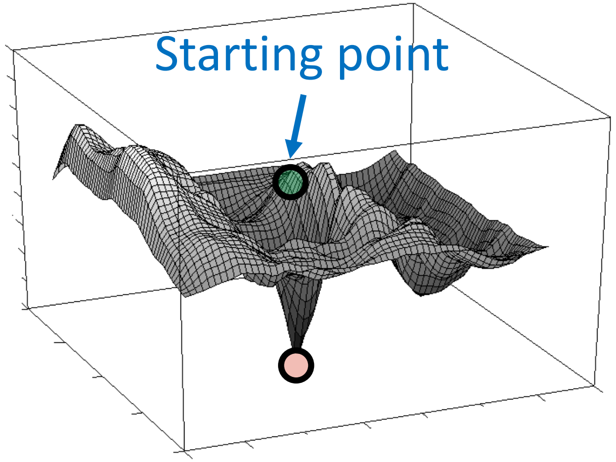
Naïve localization

- Greedy approach: calculate the SSD for all displacements and select the point where similarity is maximal – the distance is the smallest!

SSD map



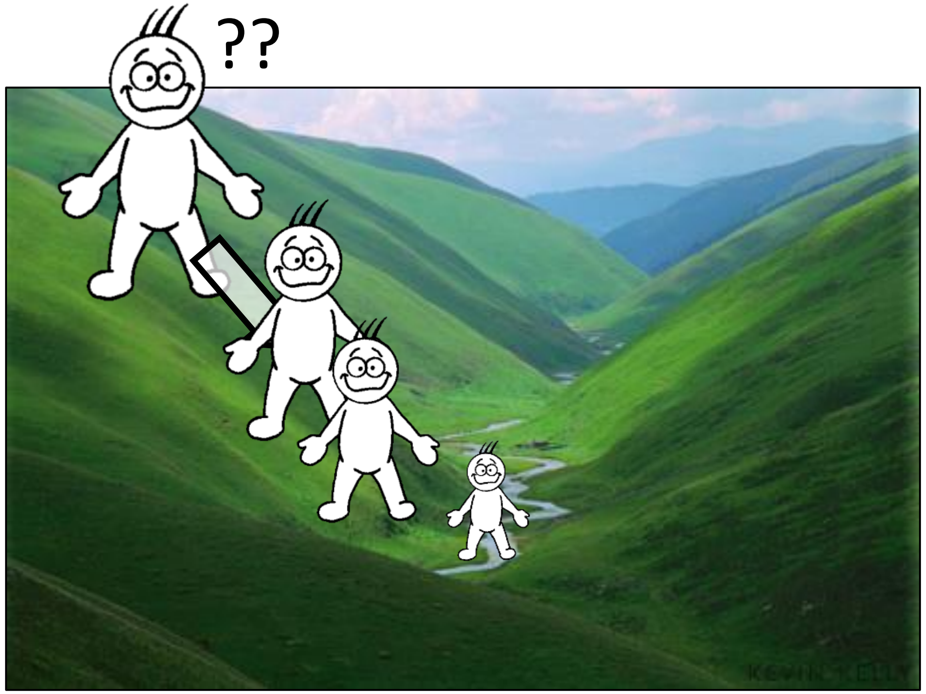
SSD map



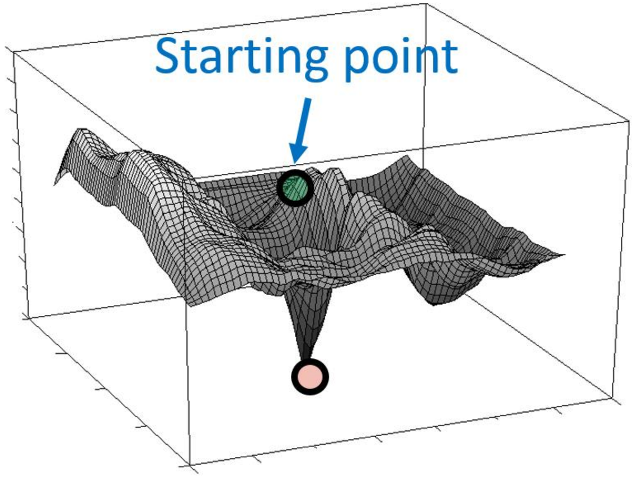
But usually we can assume our starting position is "close" to the right one!

Can we do better?

- How would we find the bottom of a valley?



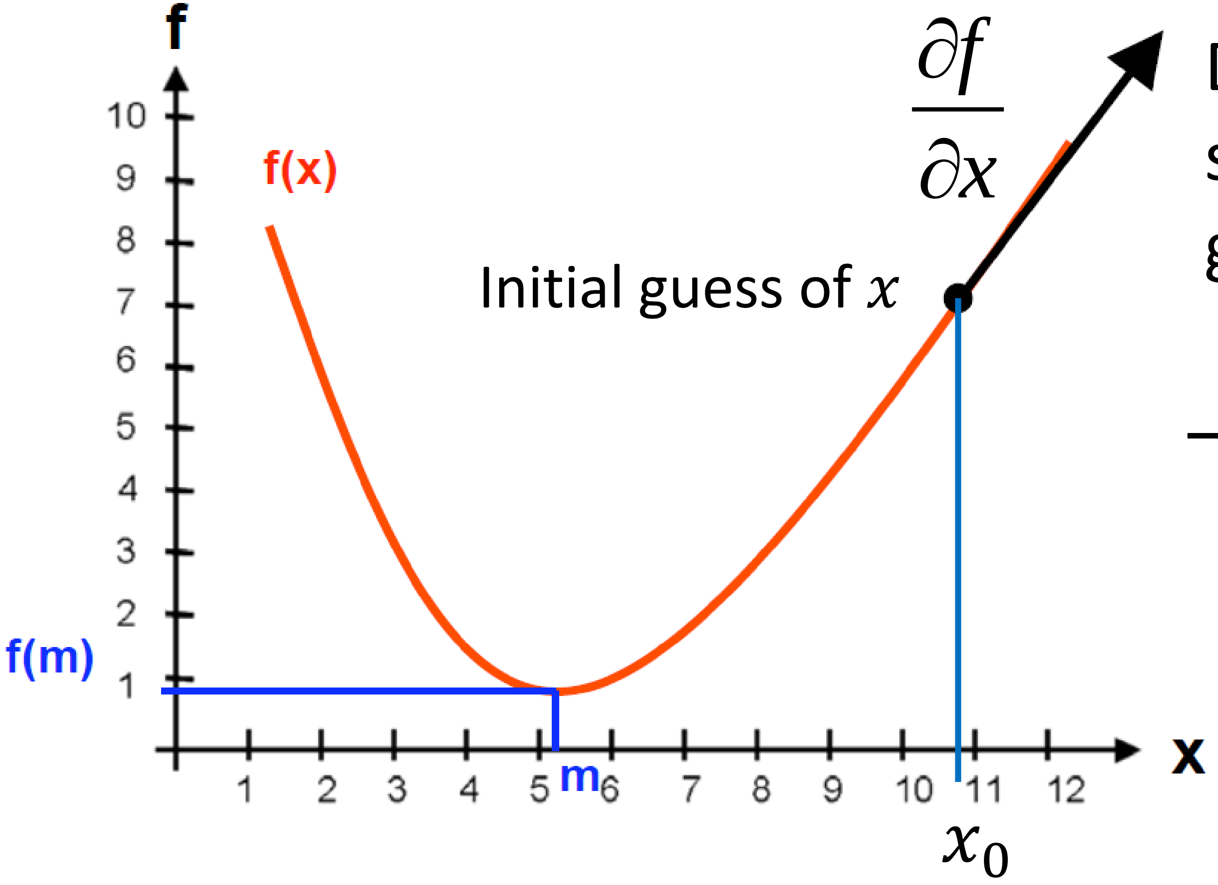
1. Decide which way is up/down
2. Move downward by some step
3. Continue until the bottom is reached



Mathematically: Known as ”the Gradient descent”

Gradient descent

- Gradient points toward *increase* of f.

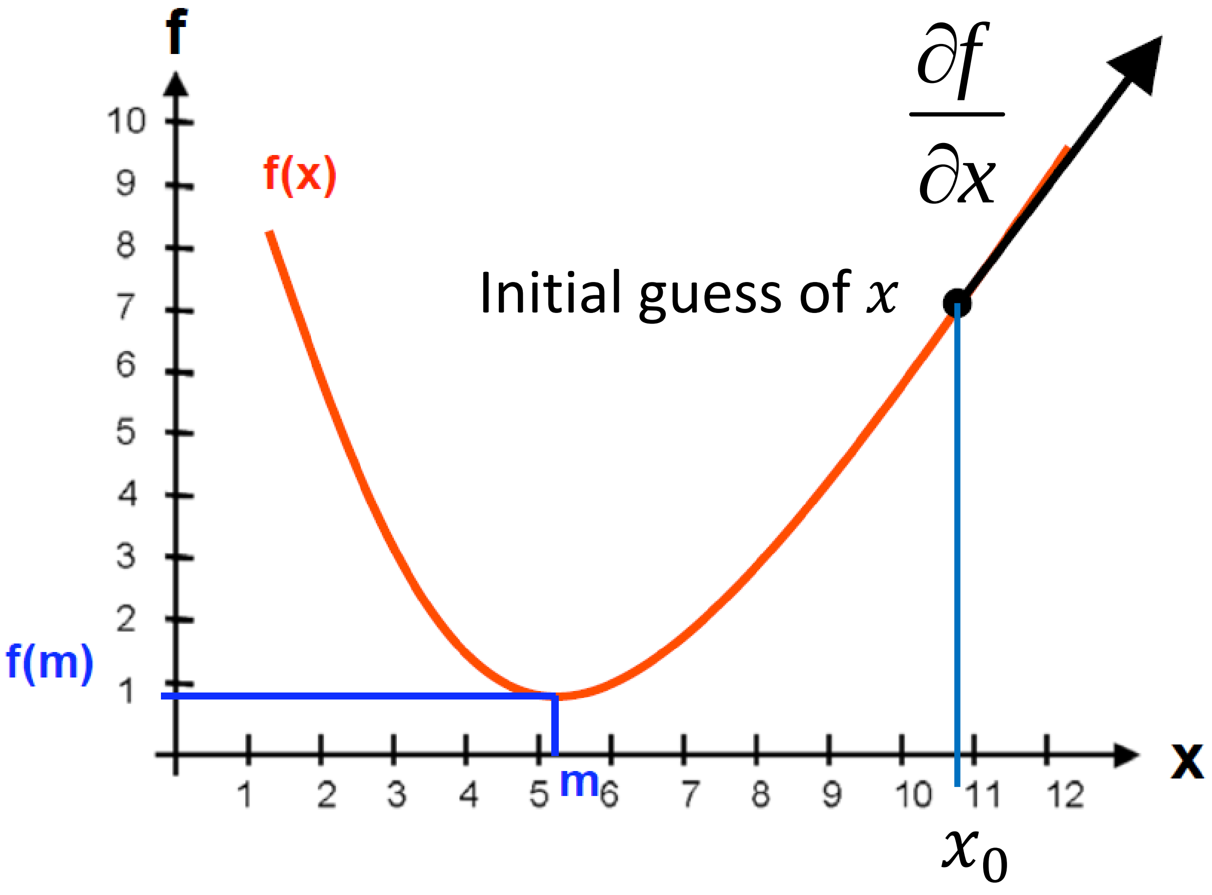


Direction in which we should be moving to get to the bottom:

$$-\frac{\partial f}{\partial x}\Big|_{x_0}$$

Gradient descent

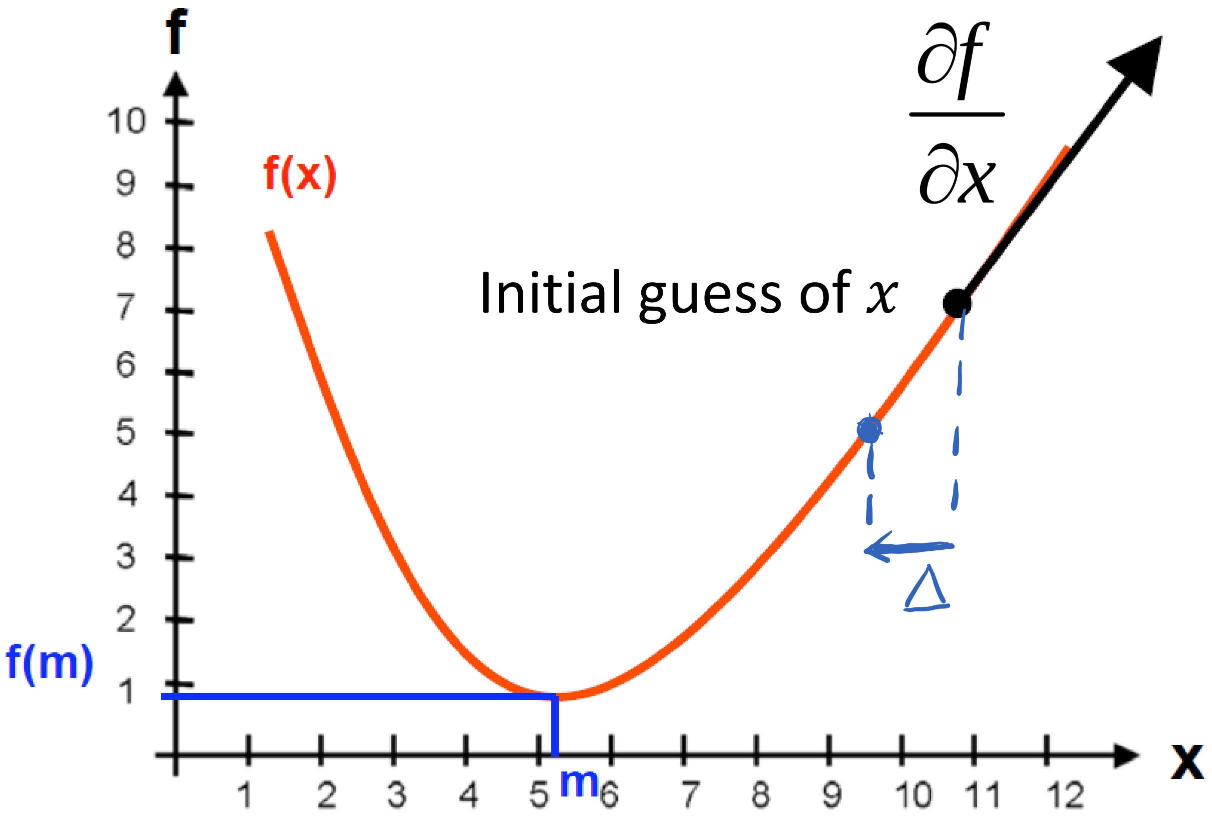
- Move in the opposite direction of the gradient



$x_1 = \mathrm{x}_0 + ?$

Gradient descent

- Move in the opposite direction of the gradient

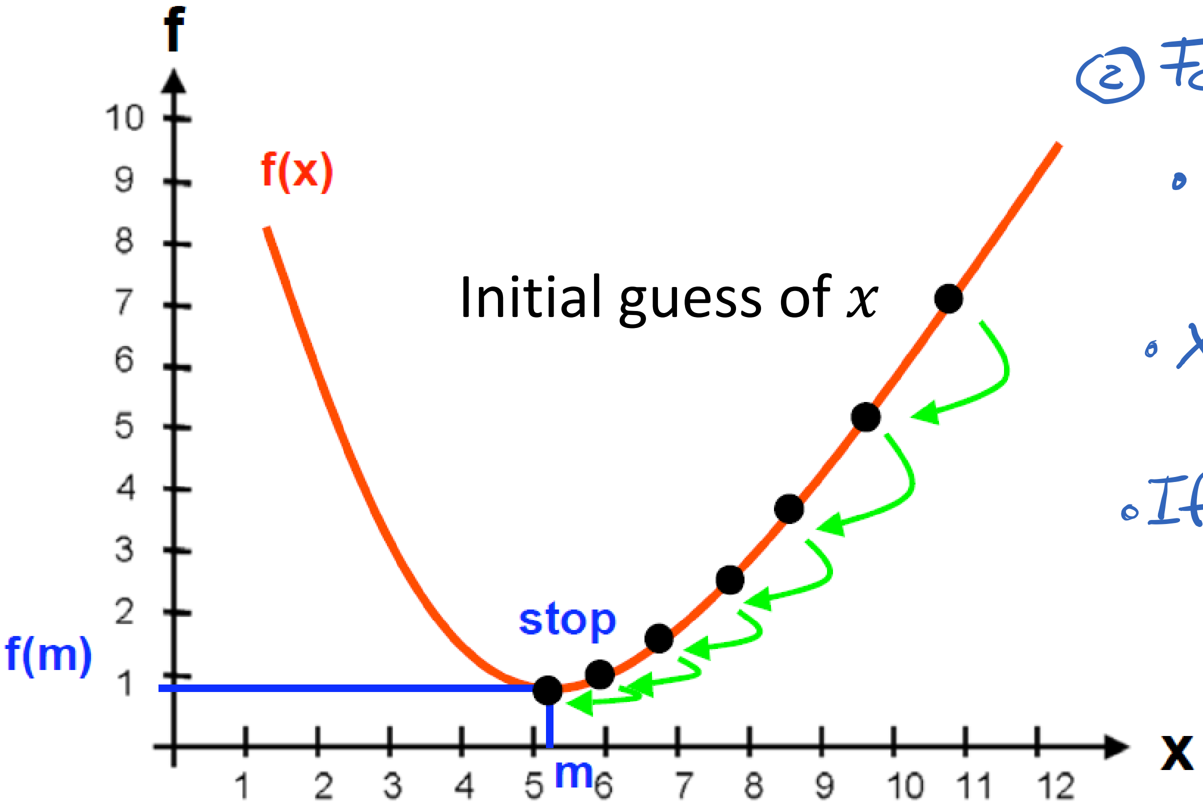


$$x_1 = x_0 - \underbrace{\left.\frac{\partial f}{\partial x}\right|_{x_0} \cdot \alpha}_{\Delta}$$

Continue with this recursion.

Gradient descent

- A simple recursive algorithm



① Start at some x_0

② For $k = 1 : \inf$

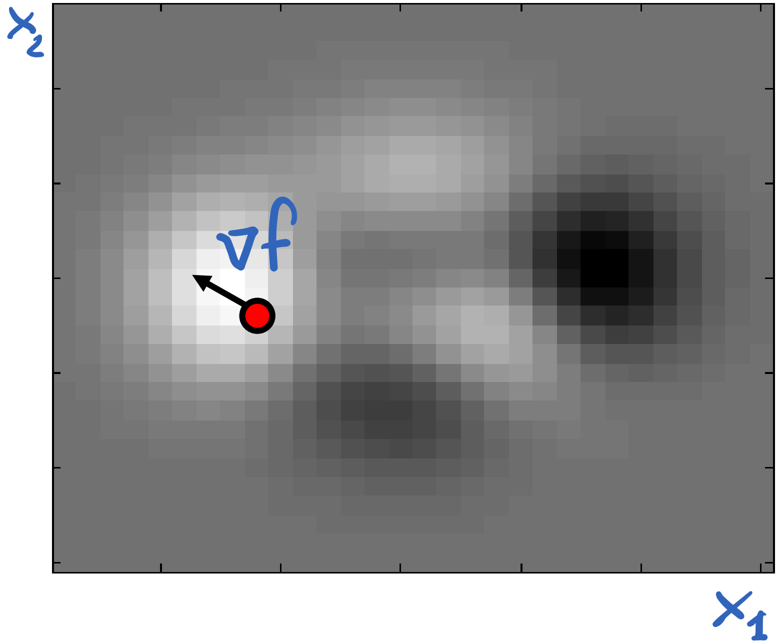
- $\Delta_k = -\left.\frac{\partial f}{\partial x}\right|_{x_{k-1}} \cdot \alpha$
- $X_k = X_{k-1} + \Delta_k$
- If $|f(x_k) - f(x_{k-1})| < \varepsilon$ exit loop

Straight-forward in n-D

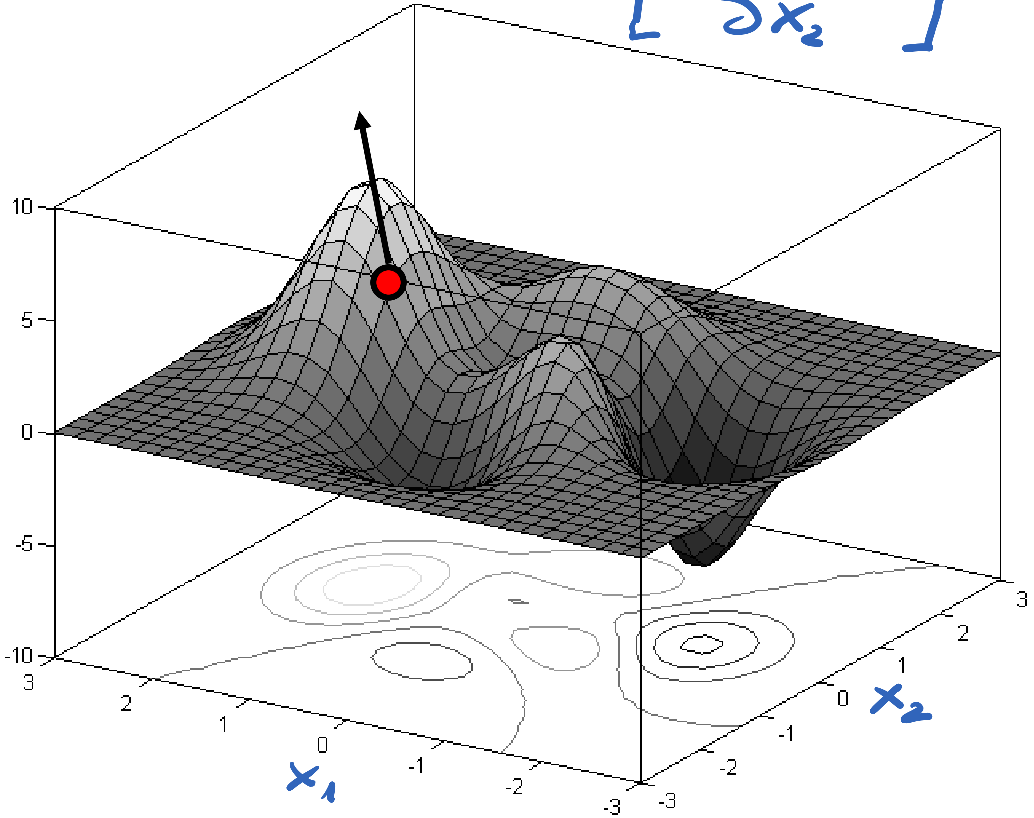
- A 2D example

$$\underline{x} = [x_1, x_2]^T$$

$$\nabla f = \frac{\partial f}{\partial \underline{x}} = \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1} \\ \frac{\partial f(x_1, x_2)}{\partial x_2} \end{bmatrix}$$



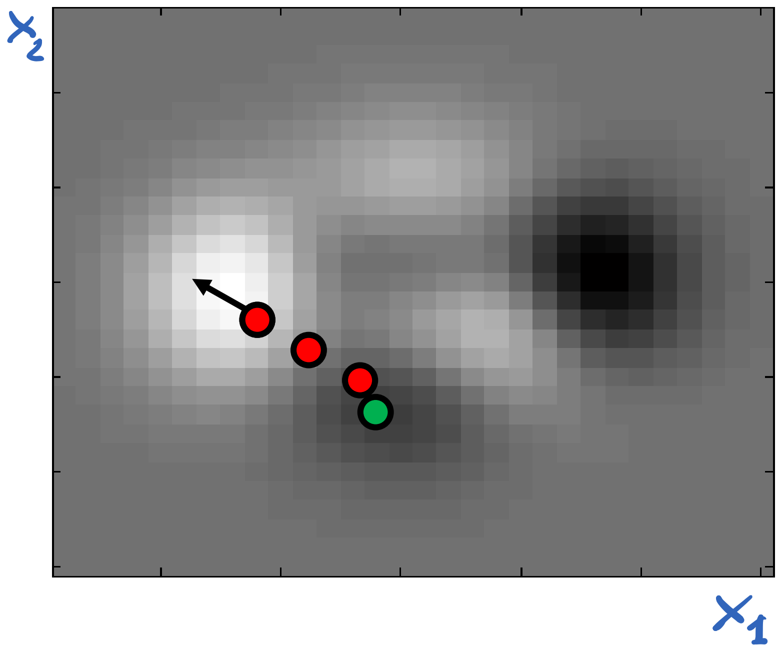
2D similarity map



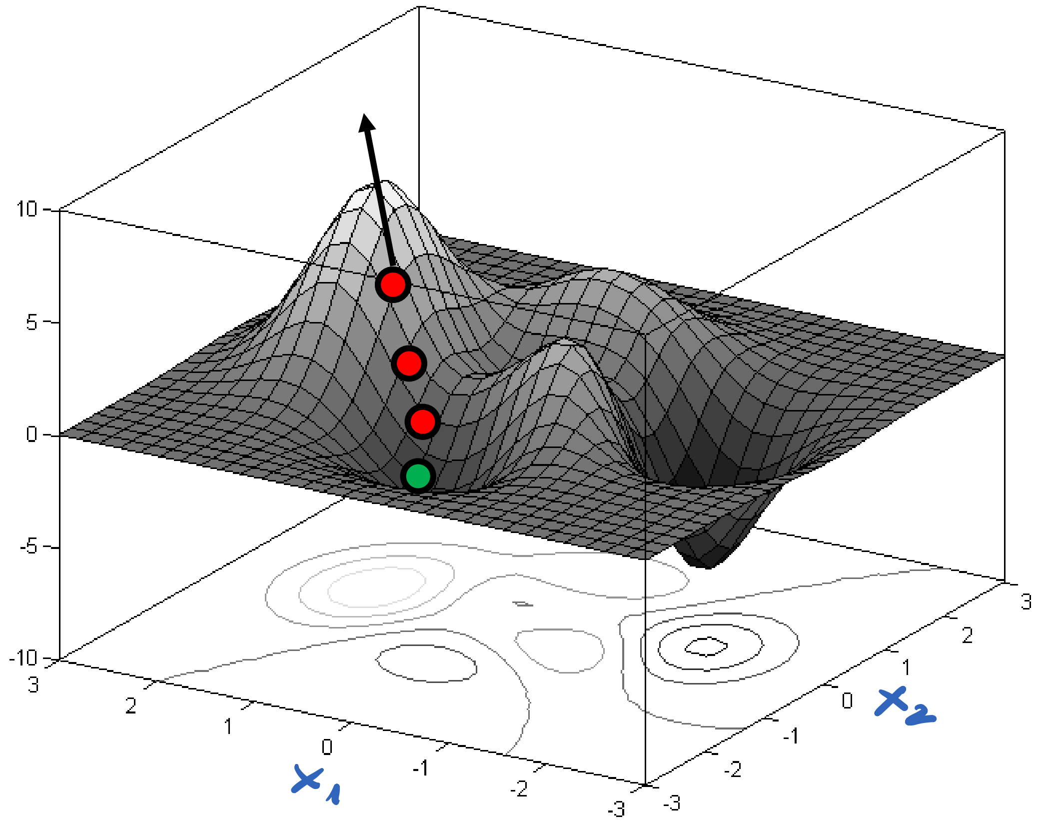
Visualize as a 3D surface

Straight-forward in n-D

- Initialize $\boldsymbol{x}_0$
- Iterate: $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} - \alpha \nabla f|_{\boldsymbol{x}_{k-1}}$



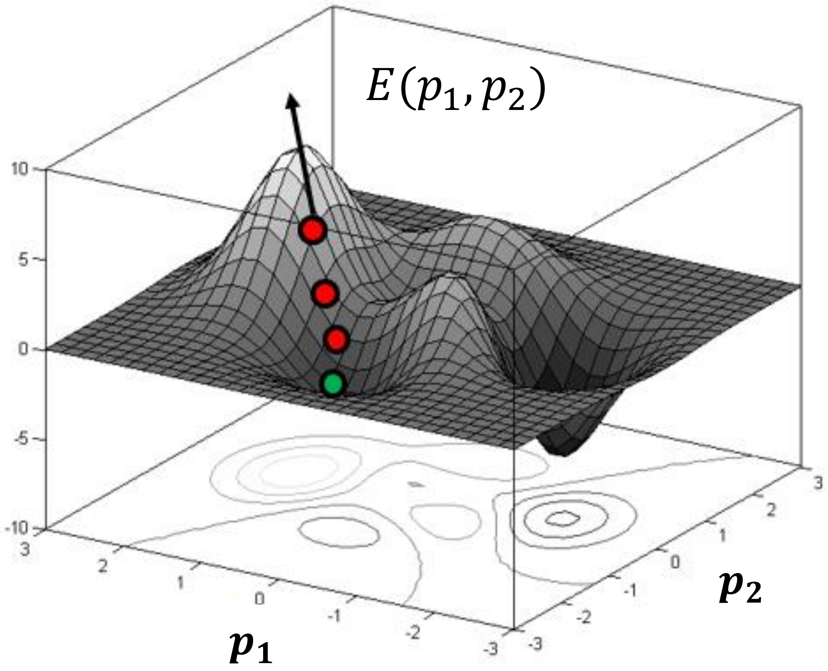
2D similarity map

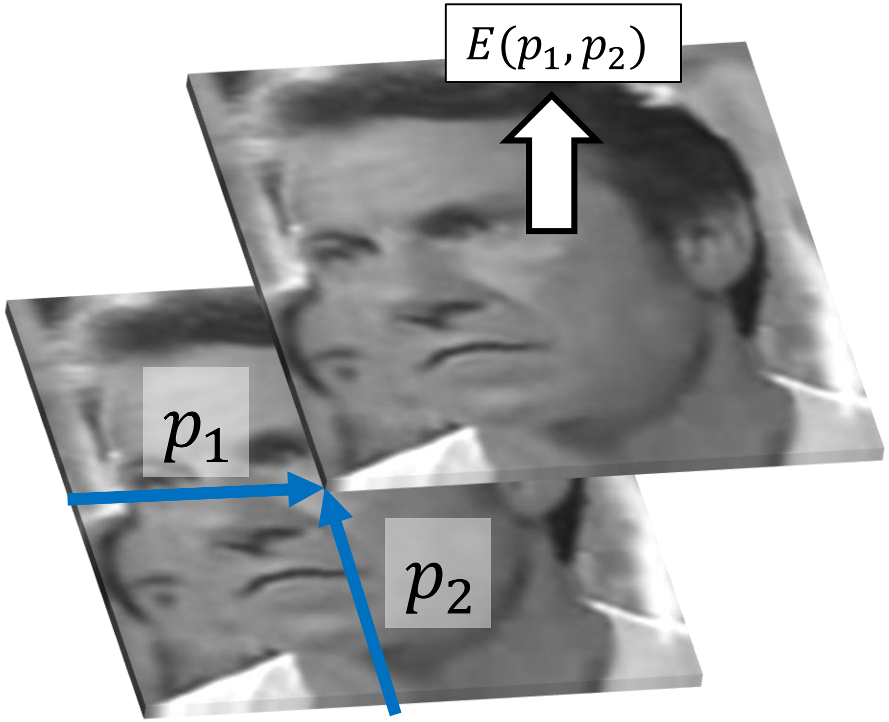


Visualize as a 3D surface

The tools we've got so far

- We know how to minimize a cost function $E(p_1, p_2, \ldots, p_N)$, w.r.t. $\boldsymbol{p}$, where $\boldsymbol{p} = [p_1, p_2, \ldots, p_N]^T$ are parameters of our model.
- We know how to compute $E(p_1, p_2)$.





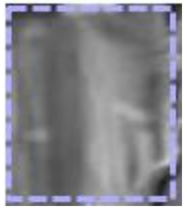
Displacement models

- Introduce a warp function *W(**x**)* that warps image onto a template – *we can think about the warp as a transformation model $W(\boldsymbol{x};\boldsymbol{p})$ that takes coordinate $\boldsymbol{x}$ and transforms it according to parameters **p**.*

$I(x)$

Example: $p = 0$

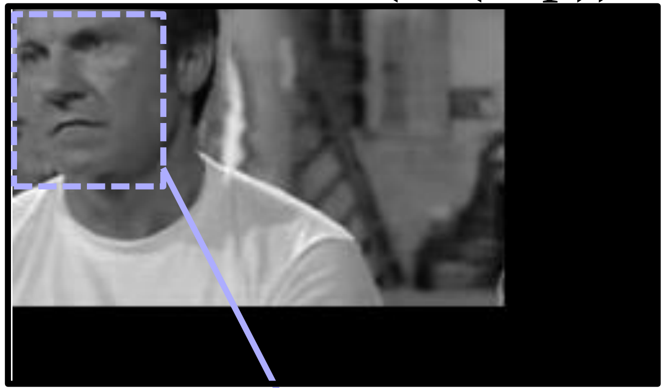
Region



$\stackrel{?}{=}$

Template

$I(\mathrm{W}(\mathrm{x};\mathrm{p}))$



Example: p … translation to left

Region

$\stackrel{?}{=}$

Template

$$\mathrm{E}(p) = \sum_{x} \left(I(\mathrm{W}(\mathrm{x};\mathrm{p})) - \mathrm{T}(\mathrm{x}) \right)^2$$

Displacement models

- Simple example:

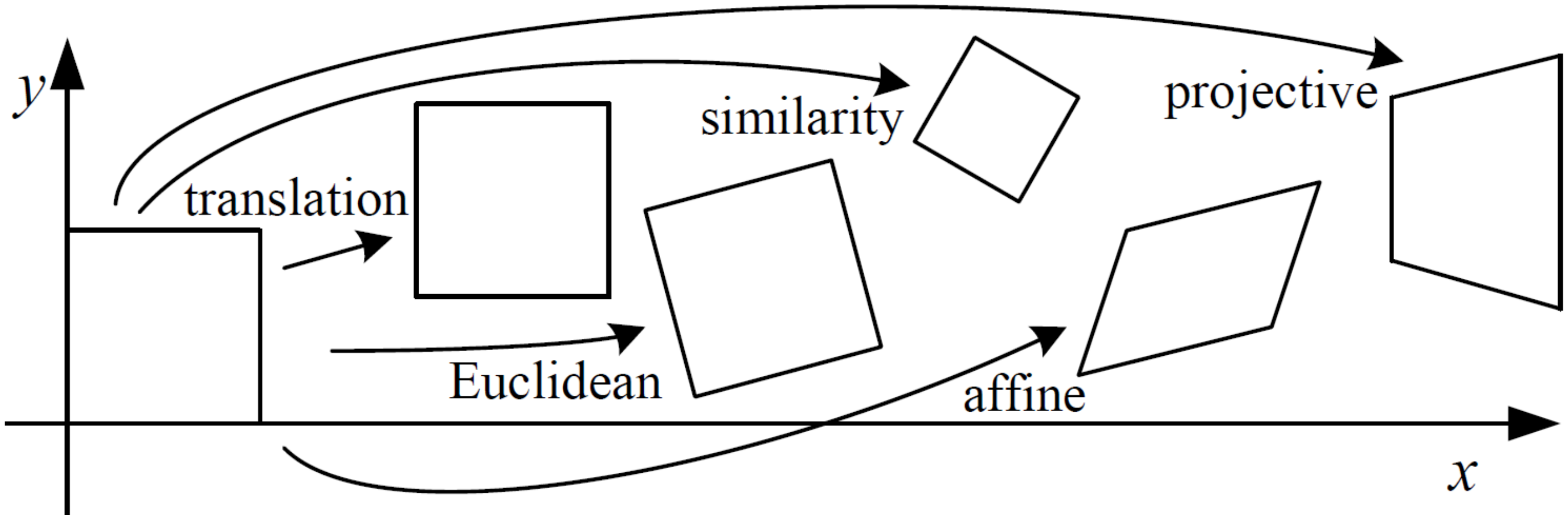
  Translation to left-up in x by 5 and y by 10.

Warped image

Original image

Displacement models

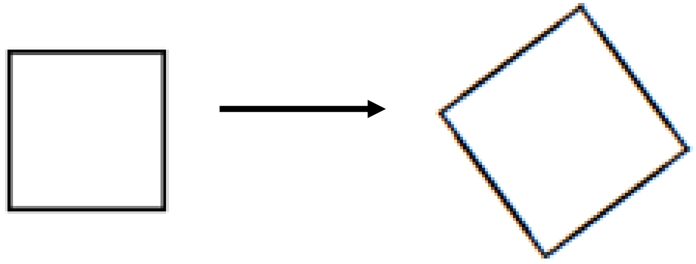
- Popular parametric 2D transformations



Richard Szeliski: Computer Vision – algorithms and applications (Section 2.1.2)

Displacement models

- Rigid body motion
  - Rotate, translate



$$x' = x\cos p_1 - y\sin p_1 + p_2$$
$$y' = x\sin p_1 + y\cos p_1 + p_3$$

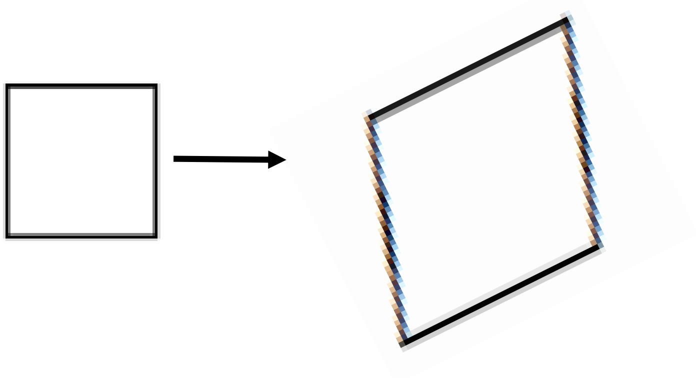
$$p = \left[p_1, p_2, p_3\right]^T$$

- Compact matrix notation for $W(\boldsymbol{x};\boldsymbol{p})$:

$$W(\mathrm{x};\mathrm{p}) = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos p_1 - y\sin p_1 + \mathrm{p}_2 \\ x\sin p_1 + y\cos p_1 + \mathrm{p}_3 \end{bmatrix} = \begin{bmatrix} \cos(\mathrm{p}_1) & -\sin(\mathrm{p}_1) & p_2 \\ \sin(\mathrm{p}_1) & \cos(\mathrm{p}_1) & p_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Displacement models

- Affine motion
  - Rotation, translation, scale, shear



$$x' = p_1 x + p_2 y + p_3$$
$$y' = p_4 x + p_5 y + p_6$$

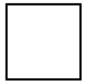
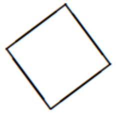
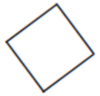
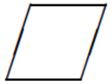
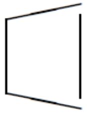
$$p = \left[p_1, p_2, p_3, p_4, p_5, p_6\right]^T$$

- Compact matrix notation for $W(\boldsymbol{x};\boldsymbol{p})$:

$$W(\mathrm{x};\mathrm{p}) = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} p_1 x + p_2 y + \mathrm{p}_3 \\ p_4 x + p_5 y + \mathrm{p}_6 \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How many free parameters?

- Degrees of freedom DoF (dim. of ***p***)

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

Richard Szeliski: Computer Vision – algorithms and applications (Section 2.1.2)

Tracking as gradient ascent/descent

- Lucas-Kanade tracker
- Initially published in 1981 as an image registration method[1].
- Improved many times, most importantly by Carlo Tomasi[2].
- Also part of the OpenCV library.
- Single algorithm and results published in a premium journal[3].
- Our derivations will follow[3]
  - See Section 2 in that paper.
  - If you're interested: *See other Sections for improvements of LK and the results obtained by these.*

[1] Lucas and Kanade. An iterative image registration technique with an application to stereo vision. ICAI, 1981.
[2] Shi and Tomasi. Good features to track. CVPR, 1994.
[3] Baker and Matthews. Lucas-Kanade 20 years on: A unifying framework. IJCV, 2004.

Lucas-Kanade algorithm

- Task: *Find the warp* $W(\boldsymbol{x};\boldsymbol{p})$ *parameterized by* $\boldsymbol{p}$*, that aligns the image* $I(\boldsymbol{x})$ *with a template* $T(\boldsymbol{x})$*.*
- For example, the warp could be a translation, i.e.,

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

  but in general $W(x;p)$ can be arbitrary.
- Problem formulation – Find the parameter values of $\boldsymbol{p}$ that minimize the image differences:

$$\mathrm{E}(p) = \sum_x \left(I(\mathrm{W}(\mathrm{x};\mathrm{p})) - \mathrm{T}(\mathrm{x}) \right)^2$$

Lucas-Kanade algorithm

$$E(p) = \sum_x \left(I(\mathrm{W}(\mathrm{x};\mathrm{p})) - \mathrm{T}(\mathrm{x}) \right)^2$$

Finding minimum of $E(\boldsymbol{p})$ w.r.t. $\boldsymbol{p}$ is a nonlinear optimization problem.

We therefore assume we have initial guess of $\boldsymbol{p}$ and search for the best increment $\Delta\boldsymbol{p}$.

$$\mathrm{E}(\mathrm{p}, \Delta p) = \sum_x \left(I(\mathrm{W}(\mathrm{x};\mathrm{p}+\Delta p)) - \mathrm{T}(\mathrm{x}) \right)^2$$

Iterative solution (think of gradient descent):

$$\mathrm{p} \leftarrow \mathrm{p} + \Delta p$$

Lucas-Kanade algorithm

- Task: Find the best $\boldsymbol{\Delta p}$: $\Delta p = \arg\min_{\Delta p} E(\mathrm{p}, \Delta p)$

$$\mathrm{E}(\mathrm{p}, \Delta p) = \sum_x \left(I(\mathrm{W}(\mathrm{x}; \mathrm{p} + \Delta p)) - \mathrm{T}(\mathrm{x}) \right)^2$$

Would have been easy if $E(p)$ was quadratic in Δp…

- To simplify, linearize $I(W(x; p + \Delta p))$ at p:

$$I\left(W\left(x; p + \Delta p\right)\right) \approx I\left(W\left(x; p\right)\right) + \nabla I^T \frac{dW}{dp} \Delta p$$

Jacobian

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}\Bigg|_{W(x,p)}$$

Note: This is a gradient of image *I* evaluated at $W(x; p)$, i.e., ∇I is computed in the coordinate frame of I and then *warped* back into the coordinate frame of T by the current estimate of the warp $W(x, p)$.

Note: In the paper of Baker&Mathews (Lucas-Kanade 20 years on…), the gradient is defined as the row vector, so the notation does not include transpose!

Jacobians of displacement models

- Translation $W(\mathrm{x};\mathrm{p}) = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x + \mathrm{p}_1 \\ y + p_2 \end{bmatrix}$

$$\frac{dW(\mathrm{x};\mathbf{p})}{d\mathbf{p}} = \begin{bmatrix} \frac{\partial \tilde{x}}{\partial p_1} & \frac{\partial \tilde{x}}{\partial p_2} \\ \frac{\partial \tilde{y}}{\partial p_1} & \frac{\partial \tilde{y}}{\partial p_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$J(\mathrm{W}) = \begin{bmatrix} \frac{\partial f_1}{\partial p_1} & \cdots & \frac{\partial f_1}{\partial p_n} \\ \frac{\partial f_2}{\partial p_1} & \cdots & \frac{\partial f_2}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial p_1} & \cdots & \frac{\partial f_m}{\partial p_n} \end{bmatrix}$$

- Affine $W(\mathrm{x};\mathrm{p}) = \begin{bmatrix} p_1 x + p_3 y + \mathrm{p}_5 \\ p_2 x + p_4 y + \mathrm{p}_6 \end{bmatrix}$

$$\frac{dW(\mathrm{x};\mathrm{p})}{dp} = \qquad ???$$

Some pre-computed Jacobians

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Richard Szeliski: Computer Vision – algorithms and applications (6.1.1.)

Lucas-Kanade algorithm

- Recall the original cost function, i.e.,

$$\mathrm{E}(\mathrm{p},\Delta p) = \sum_{x}\left(I(\mathrm{W}(\mathrm{x};\mathrm{p}+\Delta p)) - \mathrm{T}(\mathrm{x})\right)^2$$

- Plugging the linearized term into the above eq. gives

$$\mathrm{E}(\mathrm{p},\Delta p) \approx \sum_{x}\left(I\left(W\left(x;p\right)\right) + \nabla I^T \frac{dW}{d\mathbf{p}}\Delta p - \mathrm{T}(\mathrm{x})\right)^2$$

- Observe that $E(\mathrm{p},\Delta p)$ is quadratic in Δp which means that $E(\mathrm{p},\Delta \boldsymbol{p})$ can be directly minimized w.r.t. $\Delta \boldsymbol{p}$:

$$\frac{\partial\,\mathrm{E}(\mathrm{p},\Delta p)}{\partial \Delta p} \equiv 0 \qquad \Delta p = ?$$

Lucas-Kanade algorithm

Show this at home!

$$\frac{\partial \mathrm{E}(\mathrm{p}, \Delta p)}{\partial \Delta p} \equiv 0$$

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

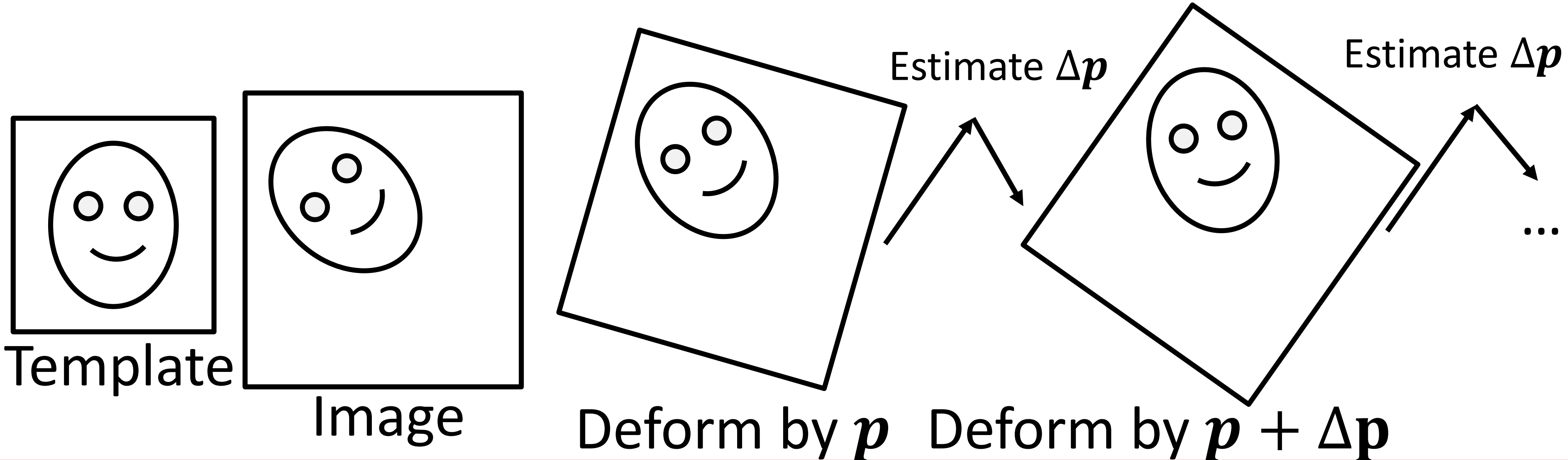
- Where ***H*** can be interpreted as a Gauss-Newton approximation of the Hessian

$$H = \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]^T \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]$$

Lucas-Kanade algorithm

Iterative solution (think of gradient descent):

- Guess initial parameters $\boldsymbol{p}$.
- Construct a linearized cost function $E(\mathrm{p}, \Delta\boldsymbol{p})$ evaluated at $\boldsymbol{p}$.
- Minimize $E(p, \Delta\boldsymbol{p})$ w.r.t. $\Delta\boldsymbol{p}$.
- Update parameters: $\mathrm{p} \leftarrow \mathrm{p} + \Delta p$



LK Implementation

Start with initial $\boldsymbol{p}$ and iterate:

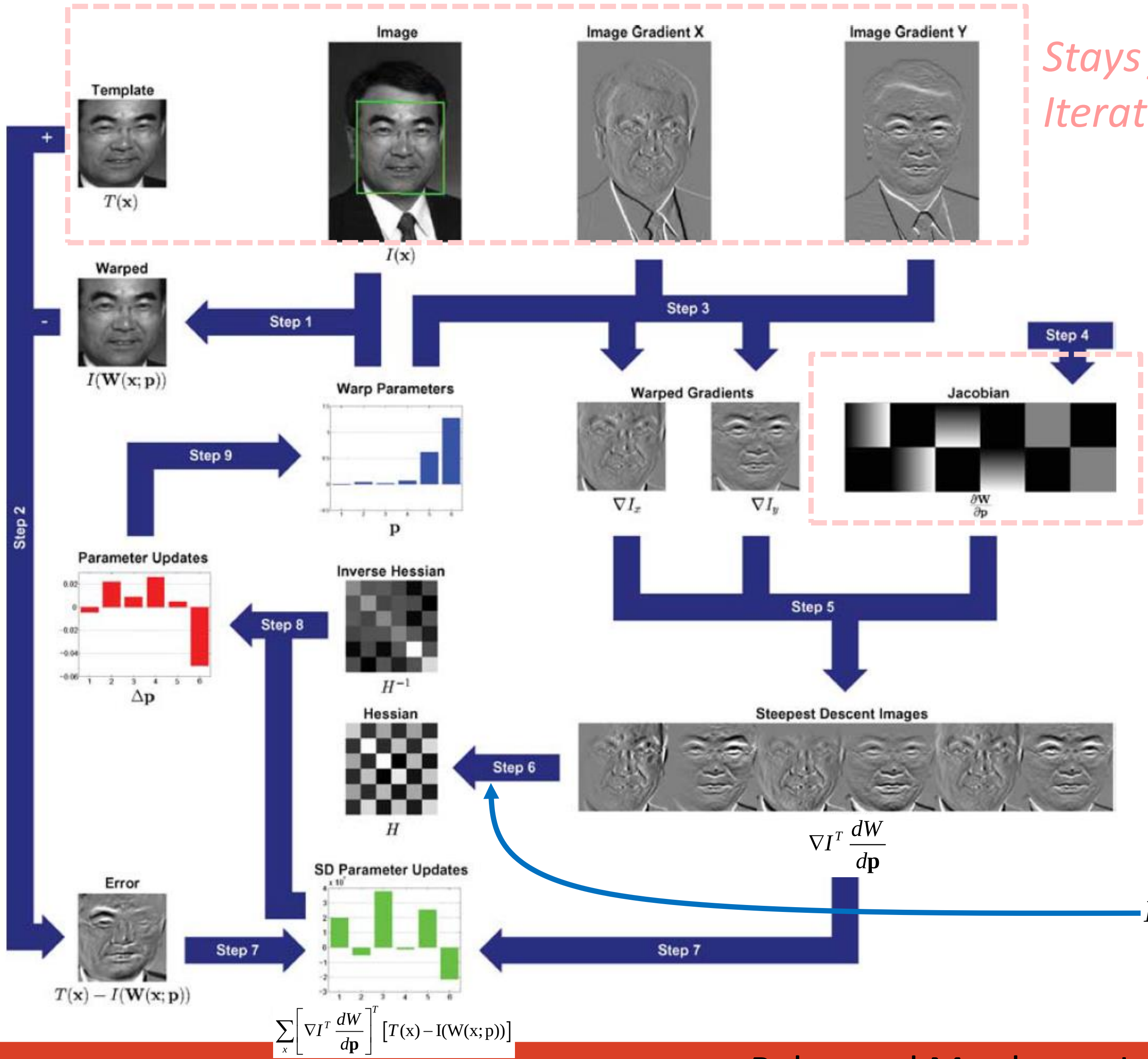
1. Warp image $I(\boldsymbol{x})$ with $W(\boldsymbol{x};\boldsymbol{p})$.
2. Warp the gradient image $\nabla I(\boldsymbol{x})$ with $W(\boldsymbol{x};\boldsymbol{p})$.
3. Evaluate the Jacobian $\frac{\partial W}{\partial \boldsymbol{p}}$ at $(\boldsymbol{x};\boldsymbol{p})$ and compute the steepest descent *image* $\nabla I^T \frac{dW}{d\boldsymbol{p}}$.
4. Compute the Hessian $H = \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}}\right]^T \left[\nabla I^T \frac{dW}{d\mathbf{p}}\right]$
5. Compute increment $\Delta p = H^{-1} \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}}\right]^T \left[T(\mathrm{x}) - \mathrm{I}(\mathrm{W}(\mathrm{x};\mathrm{p}))\right]$
6. Update parameters: $\mathrm{p} \leftarrow \mathrm{p} + \Delta p$

Until $\Delta \boldsymbol{p} < \epsilon$

(For the sake of completeness – no need to learn by heart)

LK Implementation

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



Stays fixed during Iterations.

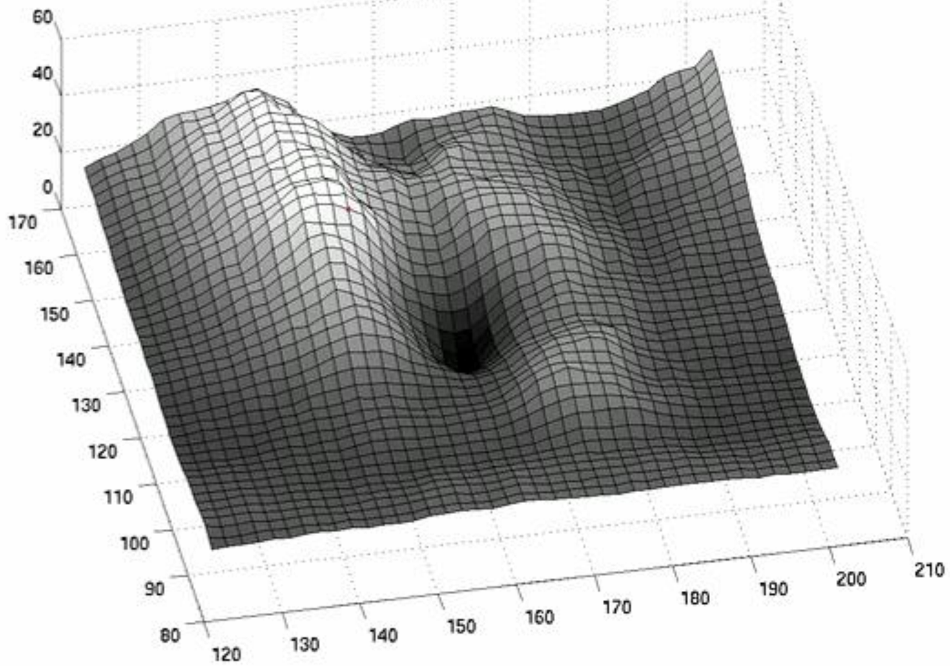
$$H = \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}}\right]^T \left[\nabla I^T \frac{dW}{d\mathbf{p}}\right]$$

Baker and Matthews. Lucas-Kanade 20 years on: A unifying framework. IJCV, 2004.

Gradient descent visualization

- Assume that warp is translation only

$$W(\mathrm{x};\mathrm{p}) = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x + \mathrm{p}_1 \\ y + p_2 \end{bmatrix}$$



Speeded up Lucas Kanade

- The original LK, spends a lot of computation on warping the image and its derivatives.

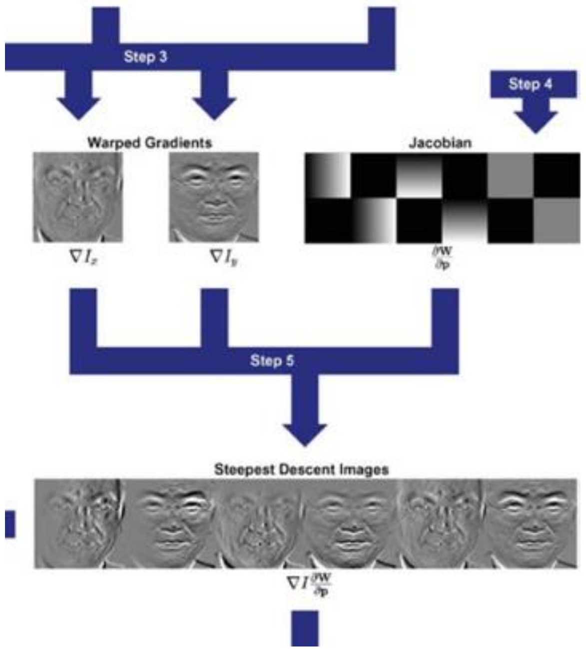
- The paper[1] suggests a simplification.

Original:

$$E(\Delta p) = \sum_x \left(I(W(x; p+\Delta p)) - T(x) \right)^2$$

New:

$$E(\Delta p) = \sum_x \left(I(W(x; p)) - T(W(x; \Delta p)) \right)^2$$



"The Inverse Compositional Algorithm" *(see paper[1], Section 3.2 for details of derivation)*

[1]Baker and Matthews. Lucas-Kanade 20 years on: A unifying framework. IJCV, 2004.

Lucas-Kanade Inverse Compositional Algorithm

Pre-compute (!!):

- Evaluate gradient ∇T of template $T(x)$.
- Evaluate Jacobian $dW/d\boldsymbol{p}$.
- Compute steepest descent images $\nabla T^T \frac{dW}{d\boldsymbol{p}}$.
- Compute hessian $H = \sum_x \left[\nabla T^T \frac{dW}{d\mathbf{p}}\right]^T \left[\nabla T^T \frac{dW}{d\mathbf{p}}\right]$

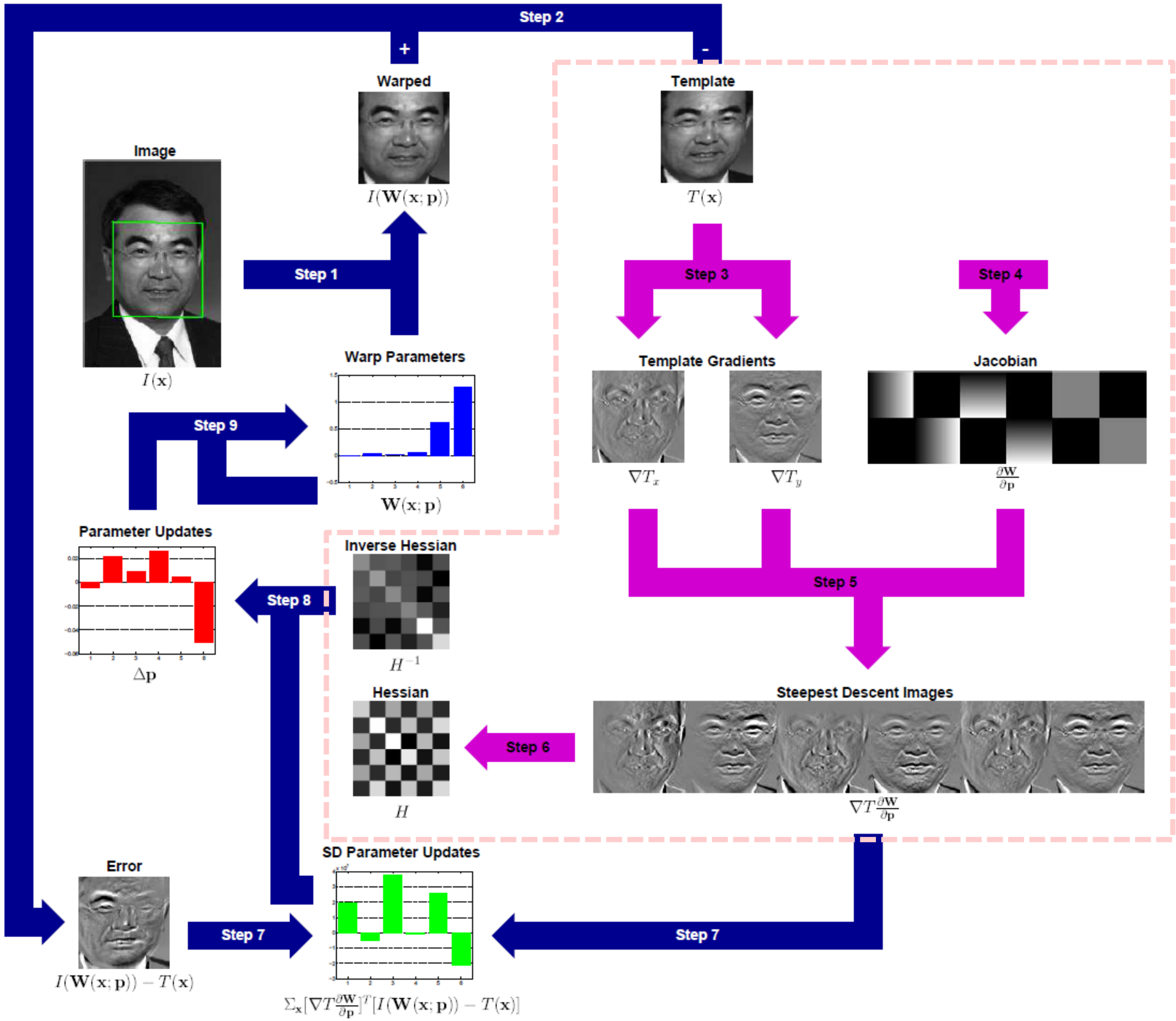
Iterate:

1. Warp image $I(\boldsymbol{x})$ with $W(\boldsymbol{x};\boldsymbol{p})$
2. Compute steepest descent $\sum_x \left[\nabla T^T \frac{dW}{d\mathbf{p}}\right]^T [\mathrm{I}(\mathrm{W}(\mathrm{x};\mathrm{p})) - T(\mathrm{x})]$
3. Compute increment $\Delta p = H^{-1} \sum_x \left[\nabla T^T \frac{dW}{d\mathbf{p}}\right]^T [\mathrm{I}(\mathrm{W}(\mathrm{x};\mathrm{p})) - T(\mathrm{x})]$
4. Update parameters $W(\mathrm{x};\mathrm{p}) \leftarrow \mathrm{W}(\mathrm{x};\mathrm{p}) \circ \mathrm{W}(\mathrm{x};\Delta p)^{-1}$

(Just for the sake of completeness – no need to learn by heart)

Lucas Kanade ICA

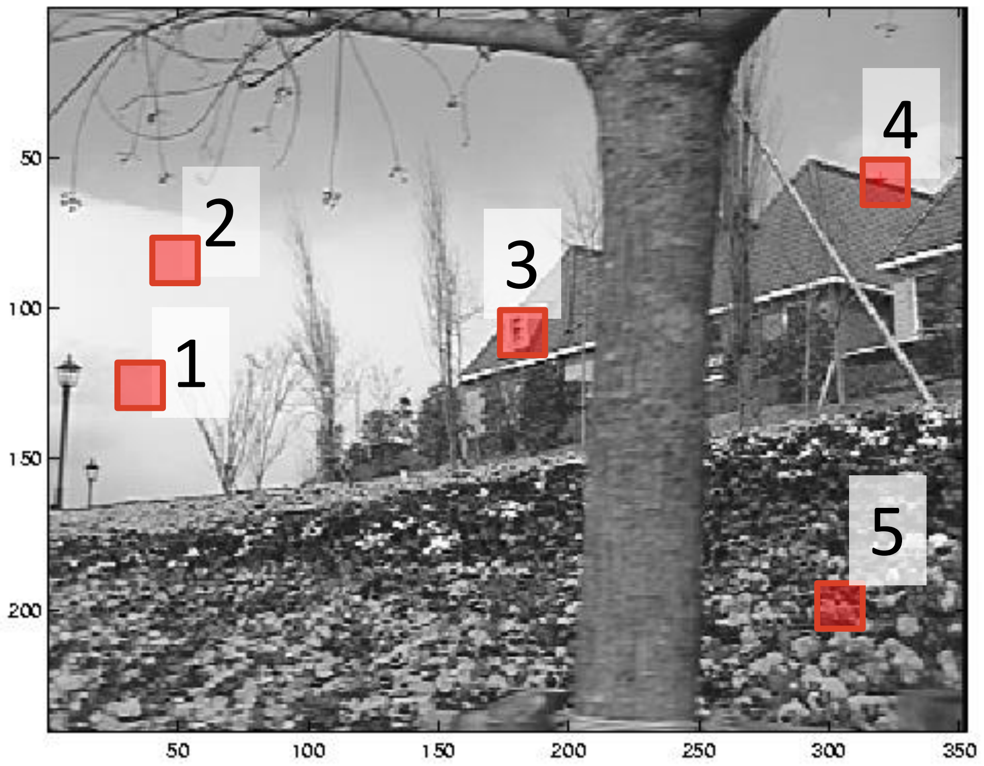
$$\Delta p = H^{-1}\sum_{x}\left[\nabla T^{T}\frac{\partial W}{\partial p}\right]^{T}[I(W(x;p)) - T(x)]$$



Stays fixed during Iterations.

What are good features to track?

- Which patches (templates) T(x) should we consider?
- Remember this discussion at LK flow estimation?



Let's look at the maths…

- Which patches (templates) T(x) should we consider?
- The ones for which we can solve the updates

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

- Stability depends on whether the Hessian is invertible

$$H = \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]^T \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]$$

What are good features to track?

- Assume that the warp function is pure translation

$$W(\mathbf{x};\mathbf{p}) = (\mathrm{x} + p_1, \mathrm{y} + \mathrm{p}_2)$$

$$\frac{dW(\mathbf{x};\mathbf{p})}{d\mathbf{p}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$H = \sum_x \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]^T \left[\nabla I^T \frac{dW}{d\mathbf{p}} \right]$$

Note that the Jacobian
is not necessarily constant in general,
but for the translational motion
it is constant!

- Then we can show that the ***H*** is in fact

$$H = \begin{bmatrix} \sum_{\mathbf{x}} I_x^{\ 2} & \sum_{\mathbf{x}} I_x I_y \\ \sum_{\mathbf{x}} I_x I_y & \sum_{\mathbf{x}} I_y^{\ 2} \end{bmatrix}$$

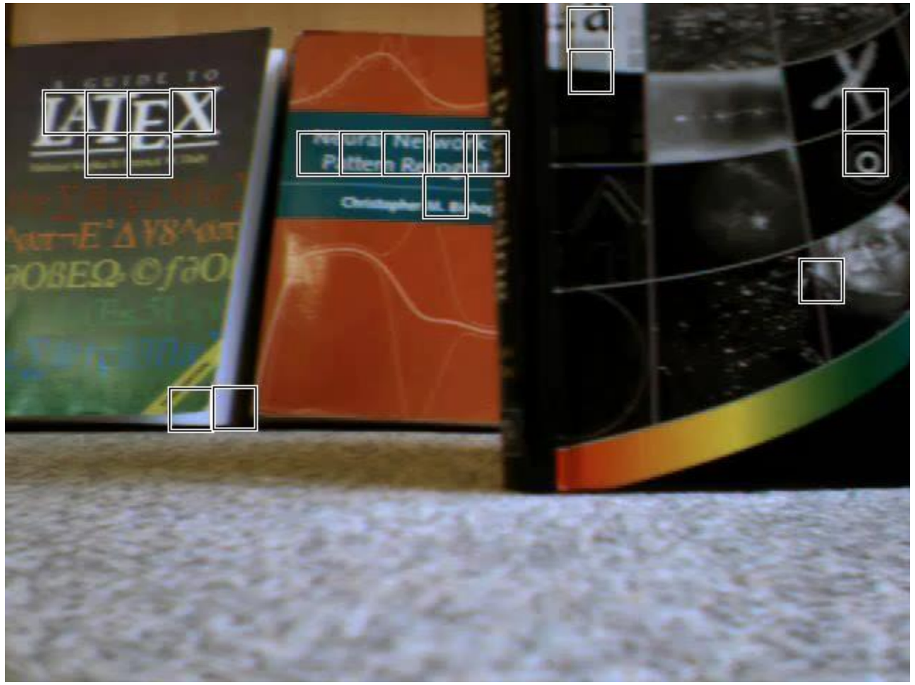
This is used in the Harris corner detector!

Verify this by yourself.

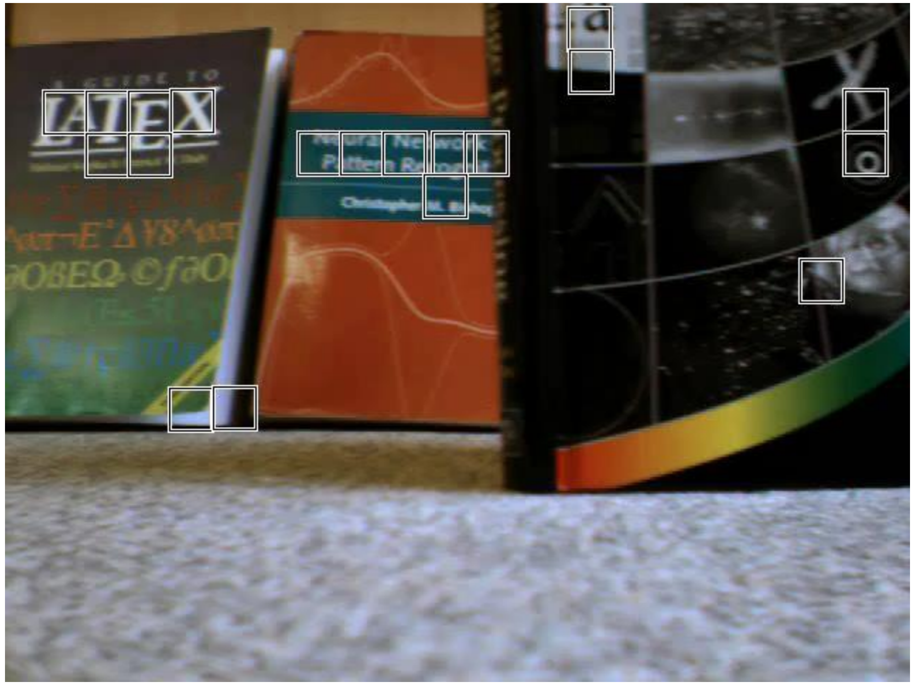
- Means that corners make good features to track.

Tracking patches

Without checking similarity with the initial patch



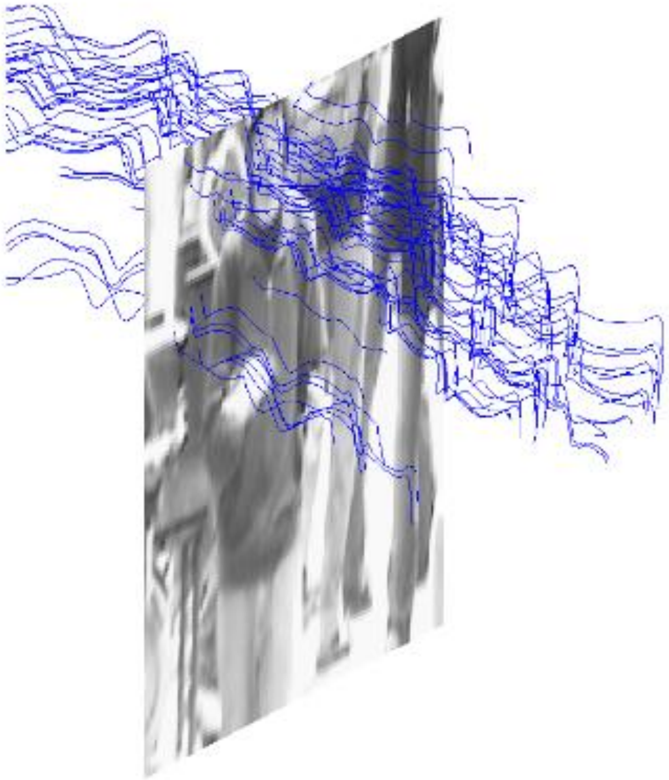
With checking similarity with the initial patch



Approach: remove a patch if similarity to initial template drops below a threshold.

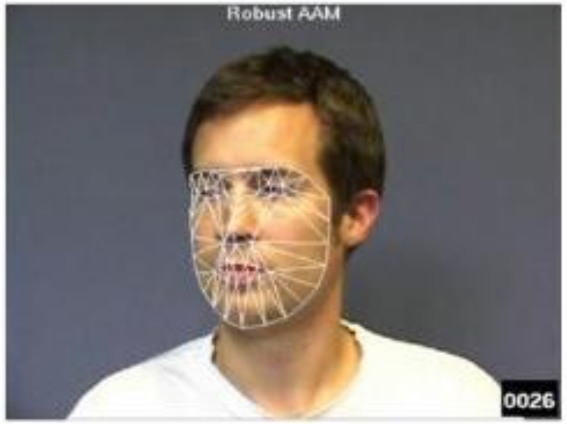
Video curtesy of Tomaš Svoboda

People counting by clustering KLT



Vincent Rabaud and Serge Belongie, Counting Crowded Moving Objects [pdf] [poster] CVPR 2006, New York, NY.

Tracking facial points by LK ICA



(a)

(d)

>200 frames per second

(c)

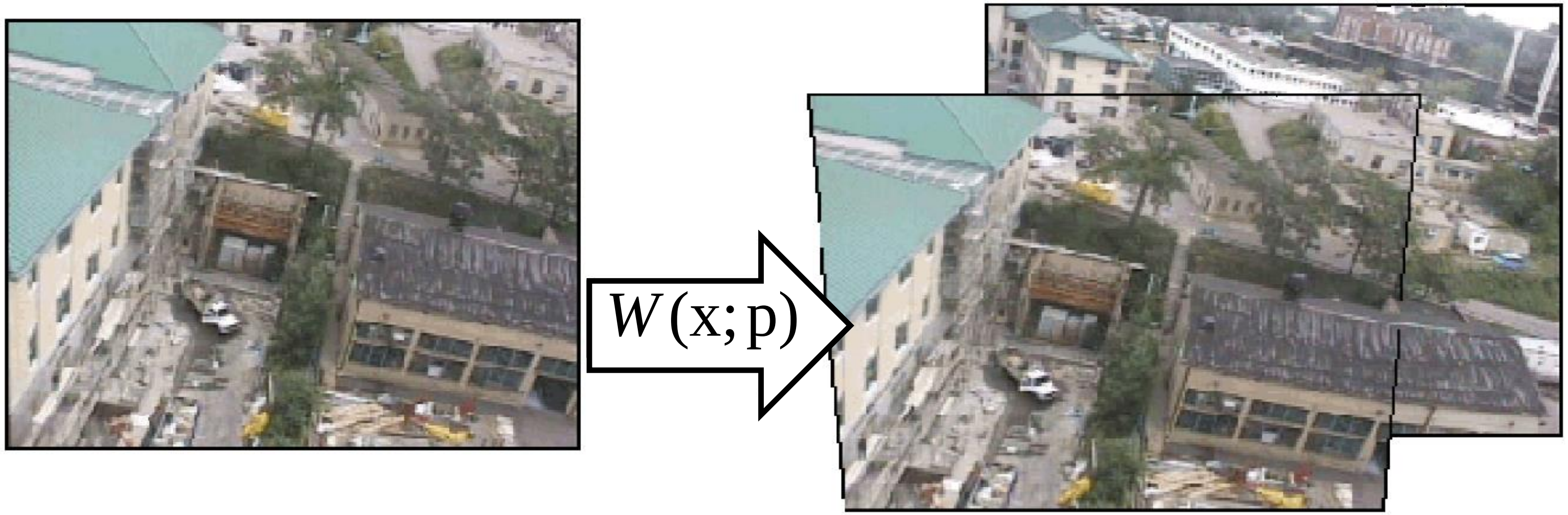
(f)

[1] Iain Matthews and Simon Baker, "Active Appearance Models Revisited," International Journal of Computer Vision, Vol. 60, No. 2, 2004

[2] Simon Baker, Iain Matthews, Jing Xiao, Ralph Gross, Takeo Kanade, and Takahiro Ishikawa, "Real-Time Non-Rigid Driver Head Tracking for Driver Mental State Estimation," 11th World Congress on Intelligent Transportation Systems, October, 2004.

Motion stabilization and stitching

- LK can be used for motion compensation
- We can consider the entire image as template

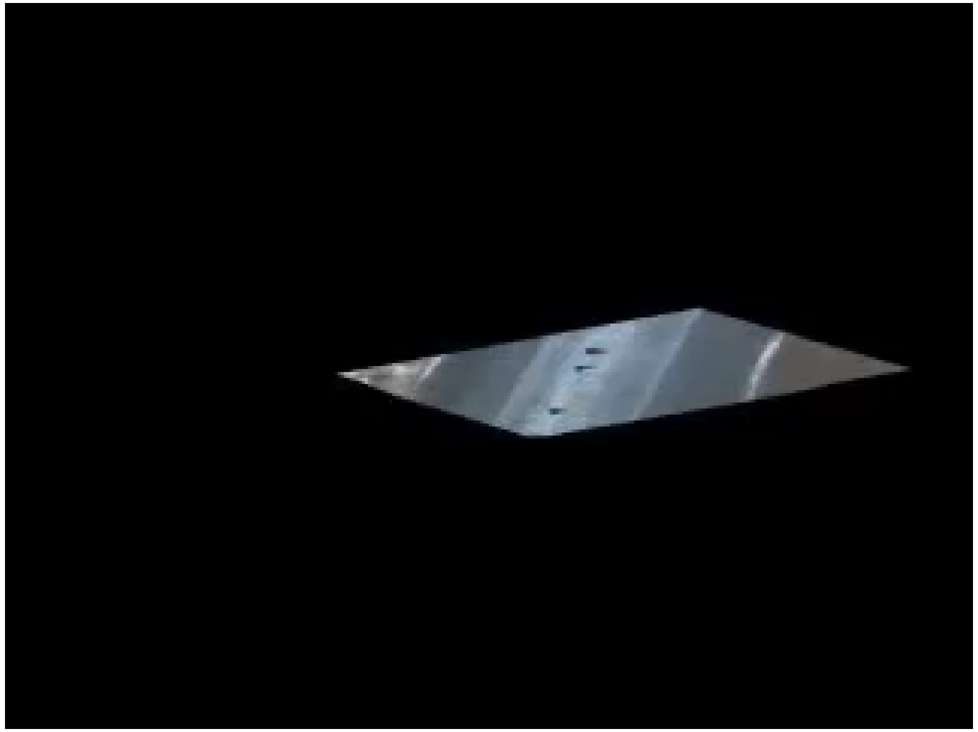


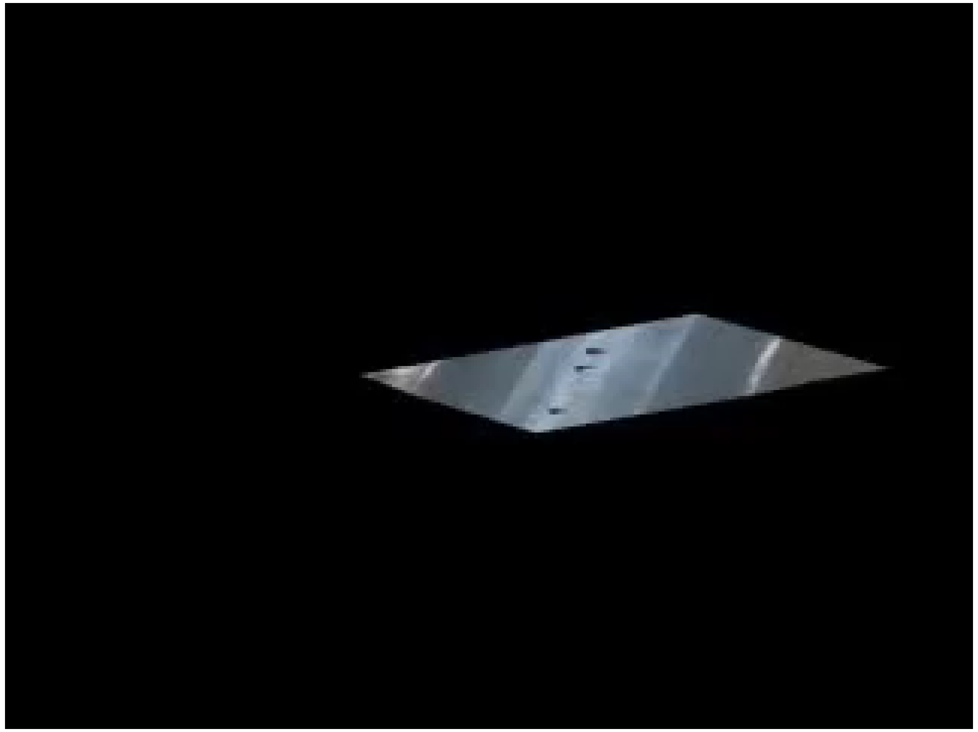
- Choose a pseudo-perspective transform for W(x;p)

  (pseudo-perspective is approximation for perspective)

Motion stabilization and stitching

- LK can be used for motion compensation
- We can consider the entire image as template





SaadAli , Mubarak Shah, COCOA -Tracking in Aerial Imagery, ISR, 2006

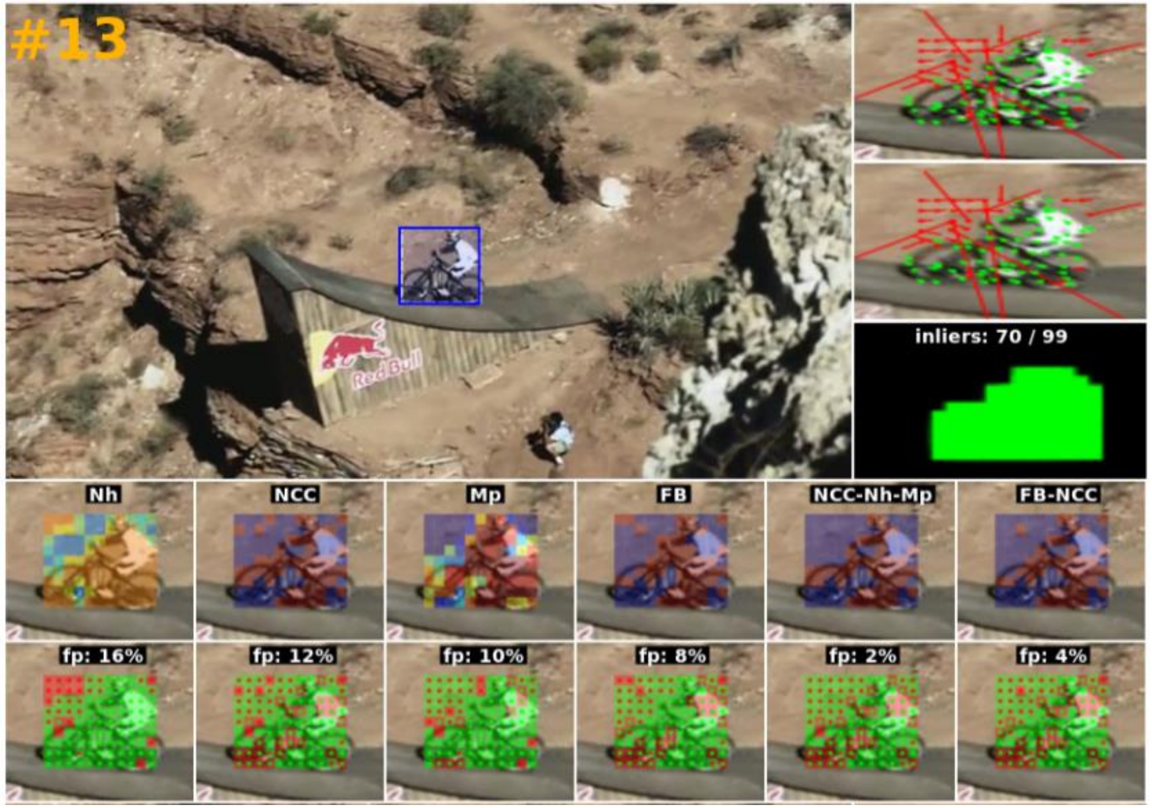
Tracking by sparse flow

- Apply Lucas Kanade (pyramidal) to estimate sparse flow.
- Fit a parametric model to the flows, e.g., affine, by least squares or RANSAC.

For least squares and RANSAC, see Richard Zseliski:
[Computer Vision – algorithms and applications]() (6.1.1-6.1.4)

Tracking by a grid of flow vectors

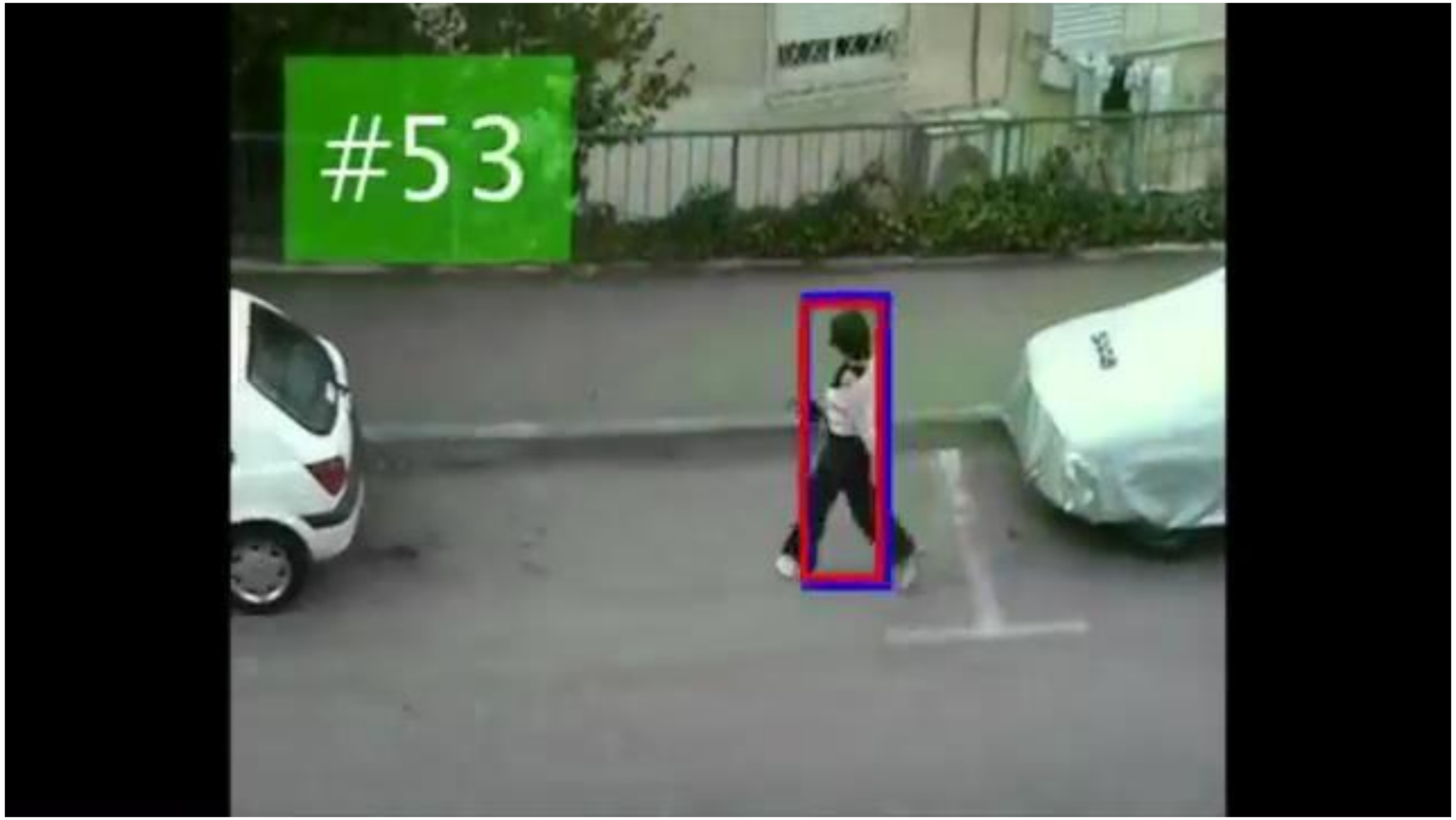
- Apply a grid of LK flows and estimate reliability of each computed flow vector.

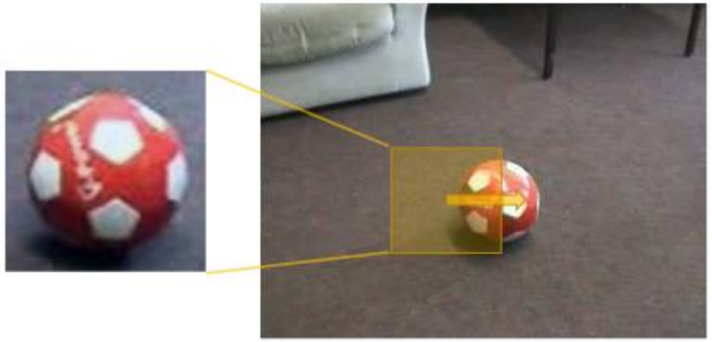


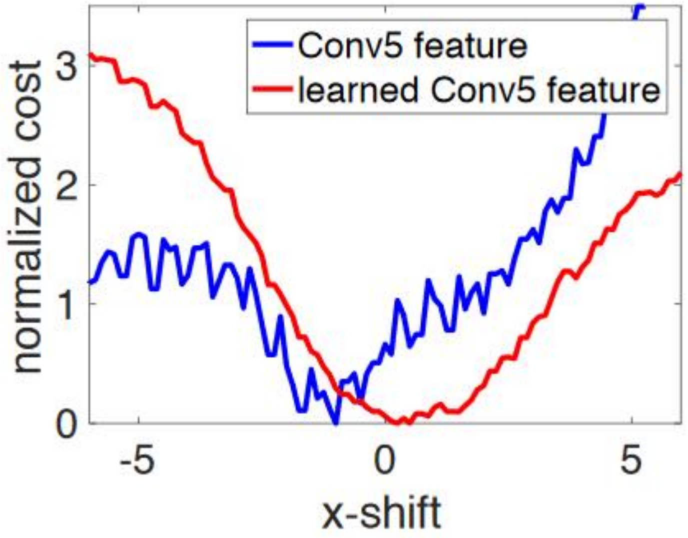
Tomas Vojir and Jiri Matas, "The Enhanced Flock of Trackers". *Registration and Recognition in Images and Videos - Studies in Computational Intelligence*, Springer 2014. (bib)

LK combination with deep features

- “Recent” work proposed learning deep features such that the cost function optimized in LK tracker becomes smooth with a large attraction perimeter







Wang et al., [Deep-LK for Efficient Adaptive Object Tracking](), ICRA2018 [[Youtube link]()]

References on LK

Recommended read:

- Baker and Matthews. Lucas-Kanade20 years on: A unifying framework. IJCV, 2004.
  - At least the section on basic Lucas&Kanade optimization

If you are interested in some milestone papers:

- Lucas and Kanade. An iterative image registration technique with an application to stereo vision. ICAI, 1981.
- Shi and Tomasi. Good features to track. CVPR, 1994.