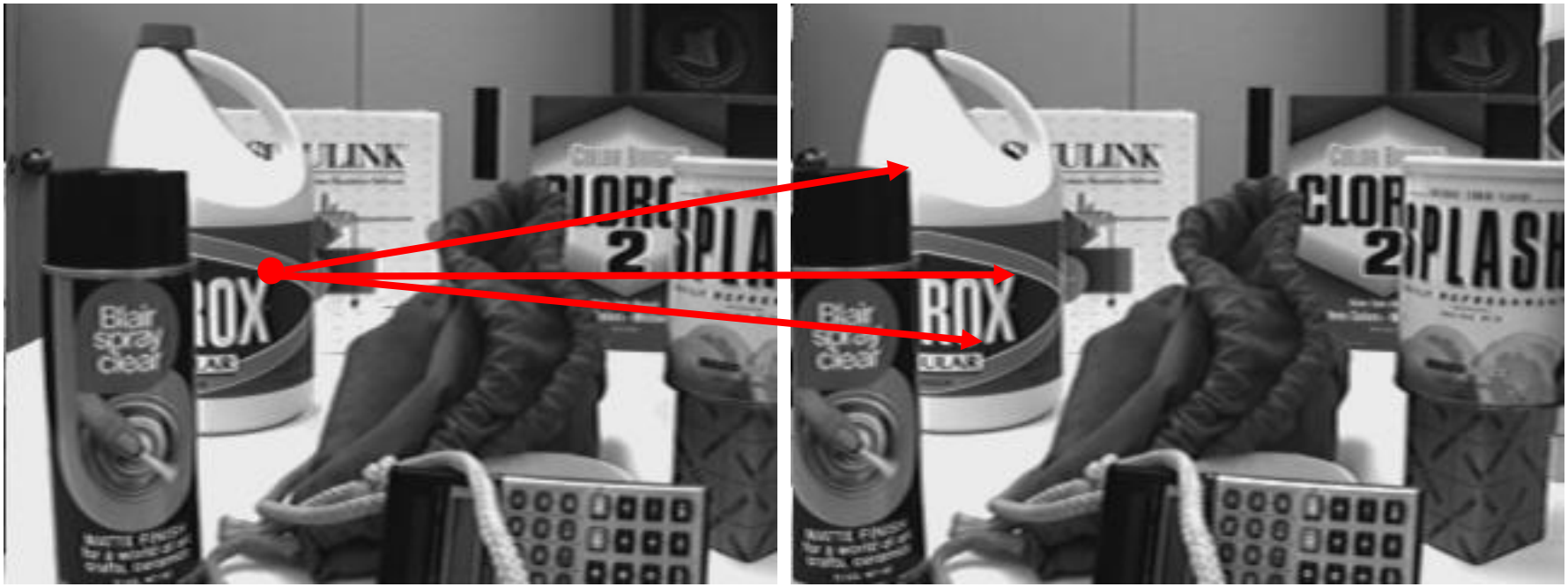

Registration tracking
(Lucas Kanade)

Video Motion estimation related areas

- Parametric motion (image alignment)
- Tracking
- Optical flow

Motion, optic flow, Tracking



Three assumptions

- Brightness consistency
- Spatial coherence
- Temporal persistence

Brightness consistency

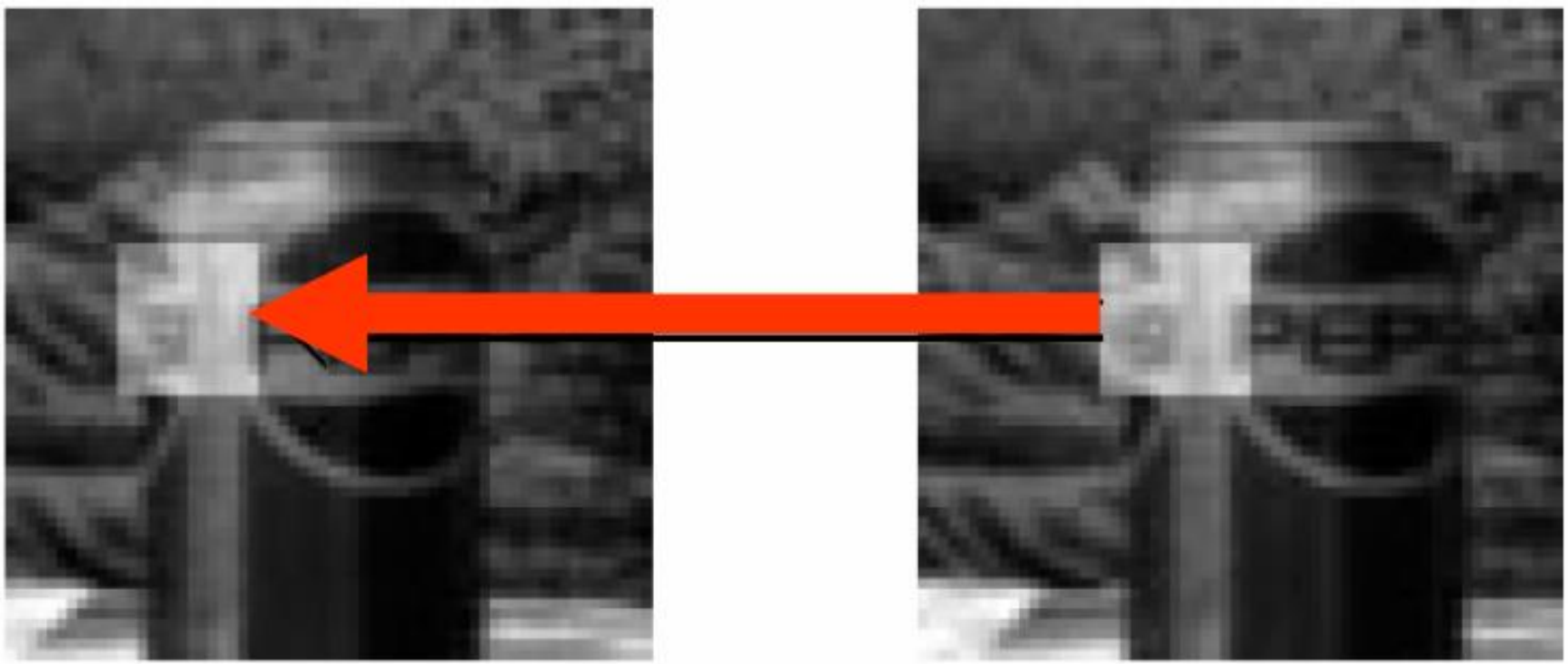
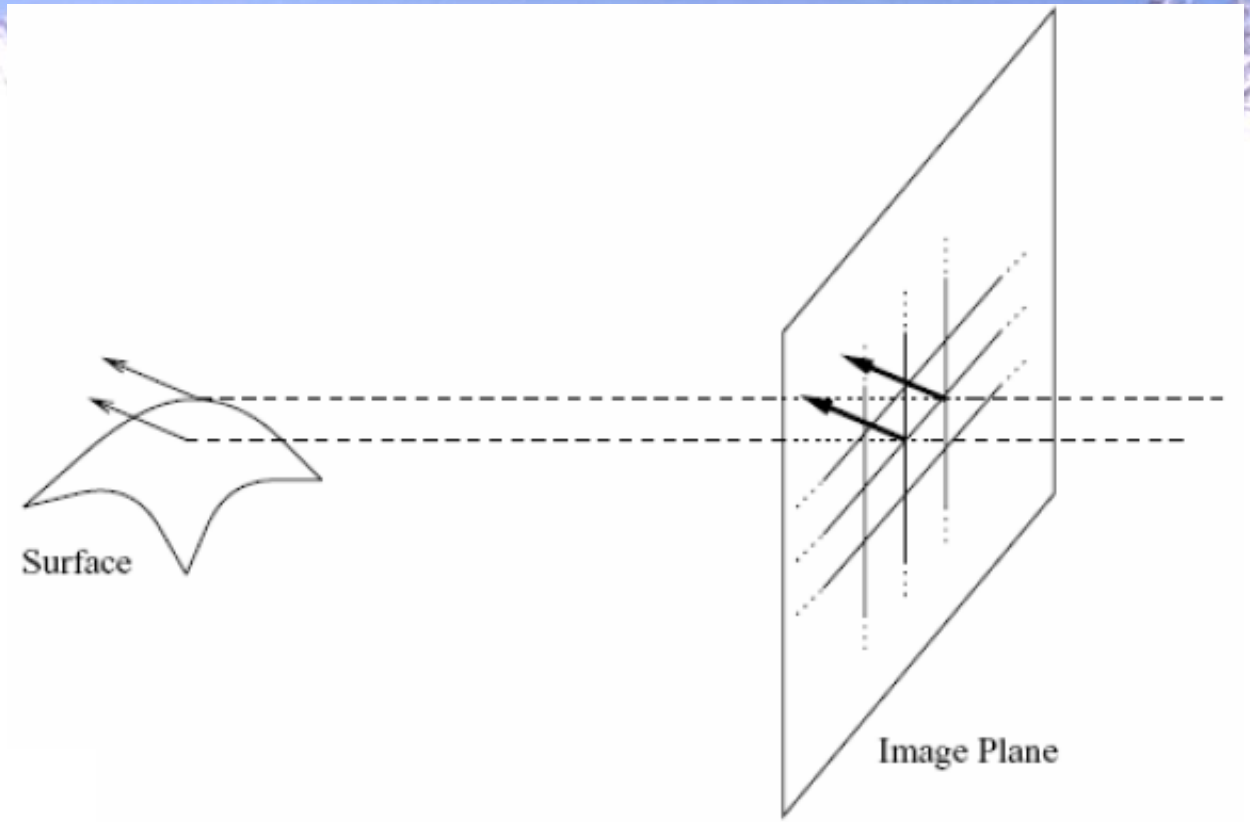


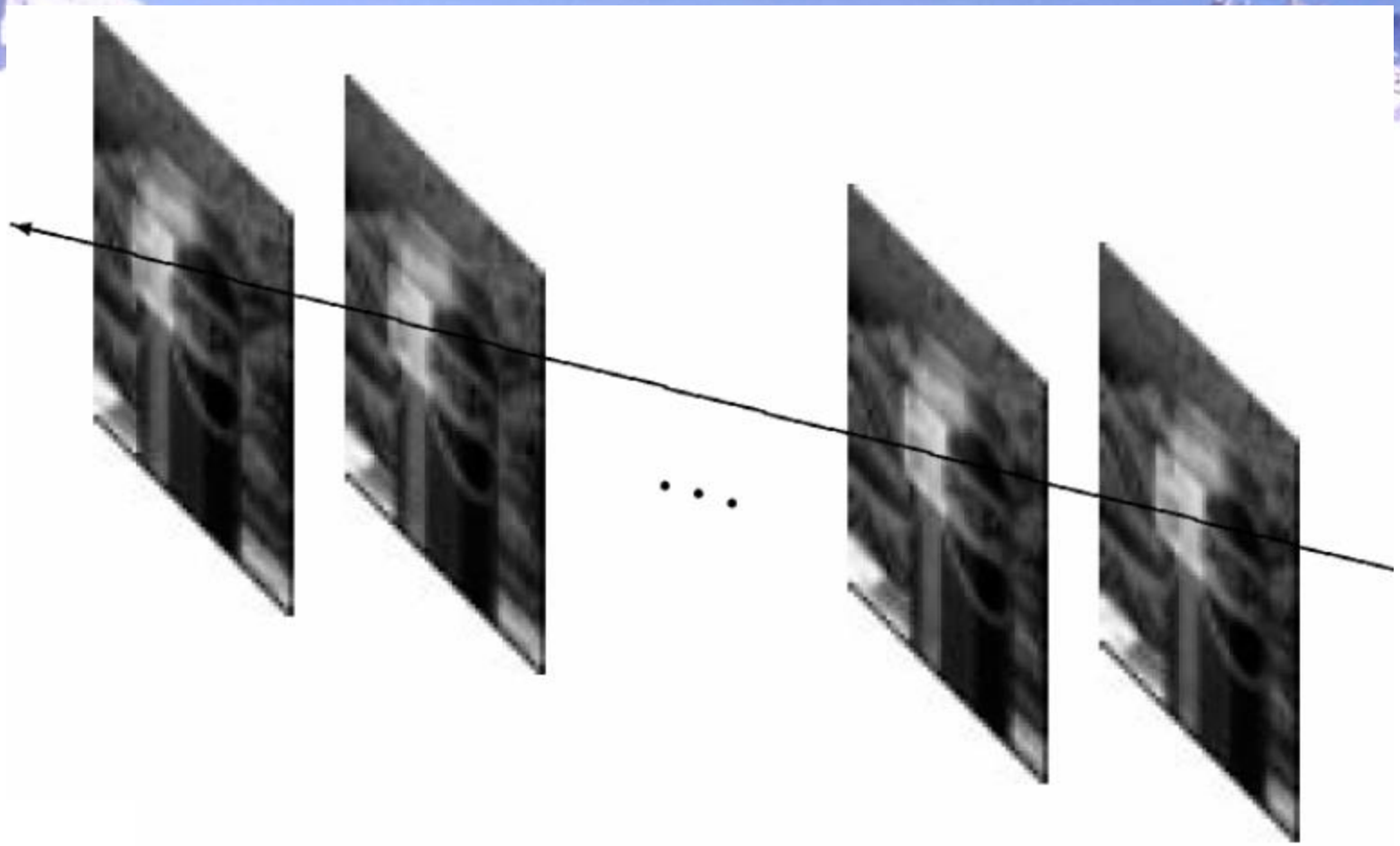
Image measurement (e.g. brightness) in a small region remain the same although their location may change.

Spatial coherence



- Neighboring points in the scene typically belong to the same surface and hence typically have similar motions.
- Since they also project to nearby pixels in the image, we expect spatial coherence in image flow.

Temporal persistence



The image motion of a surface patch changes gradually over time.

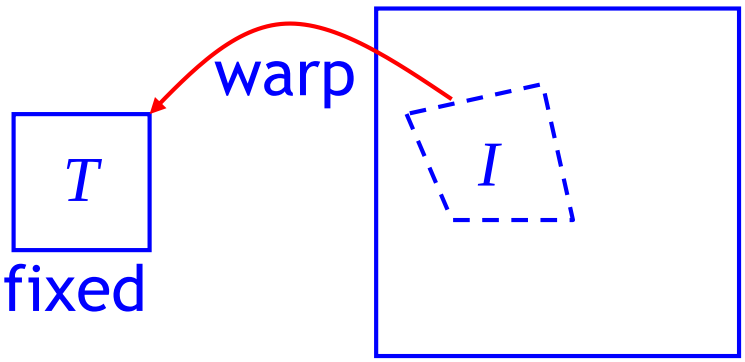
Image registration

Goal: register a template image $T(x)$ and an input image $I(x)$, where $x=(x,y)^T$. (warp I so that it matches T)

Image alignment: $I(x)$ and $T(x)$ are two images

Tracking: $T(x)$ is a small patch around a point p in the image at t. $I(x)$ is the image at time $t+1$.

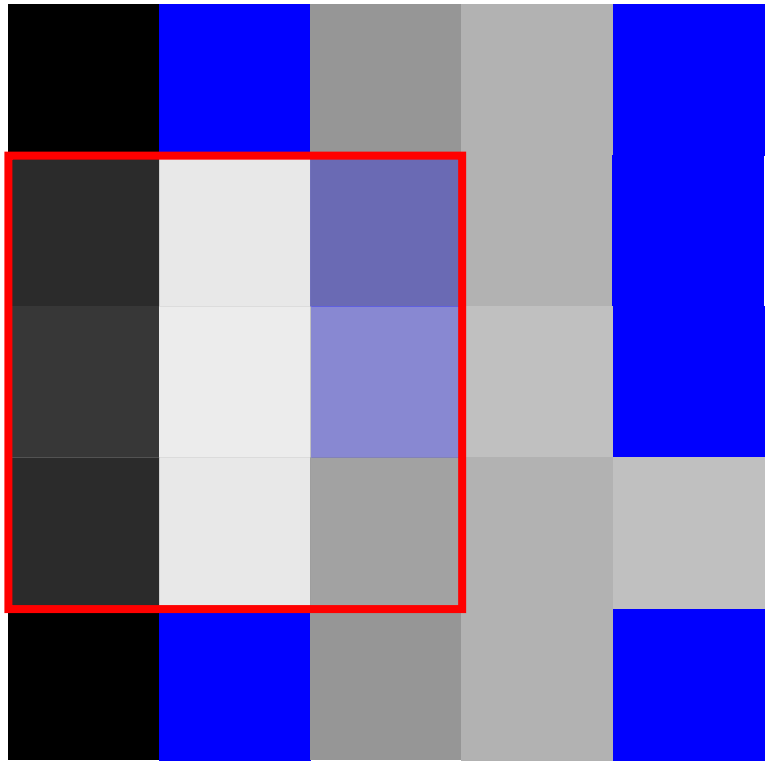
Optical flow: $T(x)$ and $I(x)$ are patches of images at t and $t+1$.

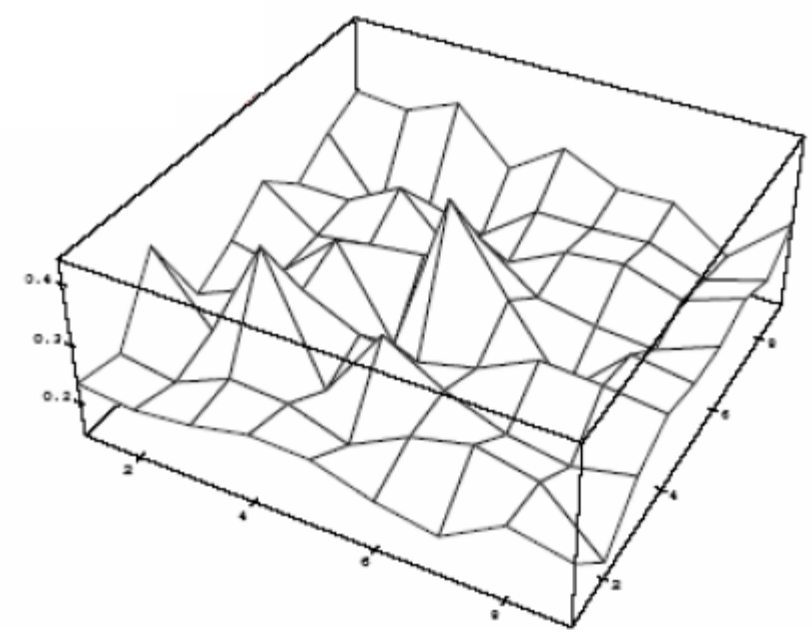


Simple approach (for translation)

- Minimize brightness difference

$$E(u,v) = \sum_{x,y} \big(I(x+u, y+v) - T(x,y)\big)^2$$





Simple SSD algorithm

For each offset *(u, v)*

compute *E(u,v);*

$$E(u,v) = \sum_{x,y} \left(I(x+u, y+v) - T(x,y) \right)^2$$

Choose *(u, v)* which minimizes *E(u,v);*

Problems:

- Not efficient
- No sub-pixel accuracy

Lucas-Kanade algorithm

$$E(u,v) = \sum_{x,y} \left(I(x+u, y+v) - T(x,y) \right)^2$$

$$I(x+u, y+v) \approx I(x,y) + uI_x + vI_y$$

$$= \sum_{x,y} \left(I(x,y) - T(x,y) + uI_x + vI_y \right)^2$$

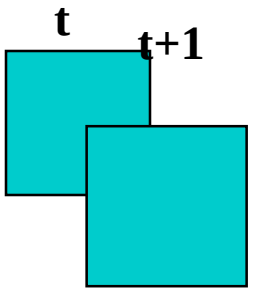
$$= \sum_{x,y} \left(d\,\mathrm{Im}\_t(x,y) + uI_x + vI_y \right)^2$$

Optic Flow Lucas-Kanade algorithm

- Solving for the translational motion of a patch

Over determined equation system:

$$\begin{pmatrix} \vdots \\ -\frac{\partial \mathrm{Im}}{\partial t} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \vdots \\ \frac{\partial \mathrm{Im}}{\partial x} & \frac{\partial \mathrm{Im}}{\partial y} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix}$$



$$-\mathrm{Im\_t} = \mathrm{M}\,\mathrm{u}$$

- Can be solved in least squares using matlab

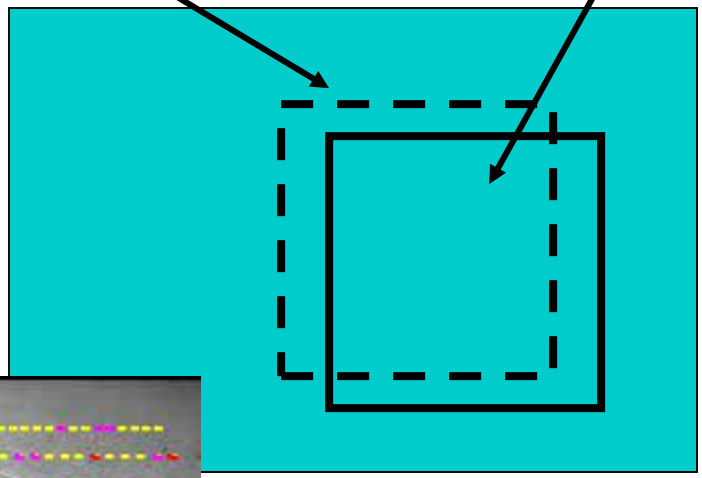
$$\mathrm{u} = \mathrm{M}\backslash\mathrm{Im\_t}$$

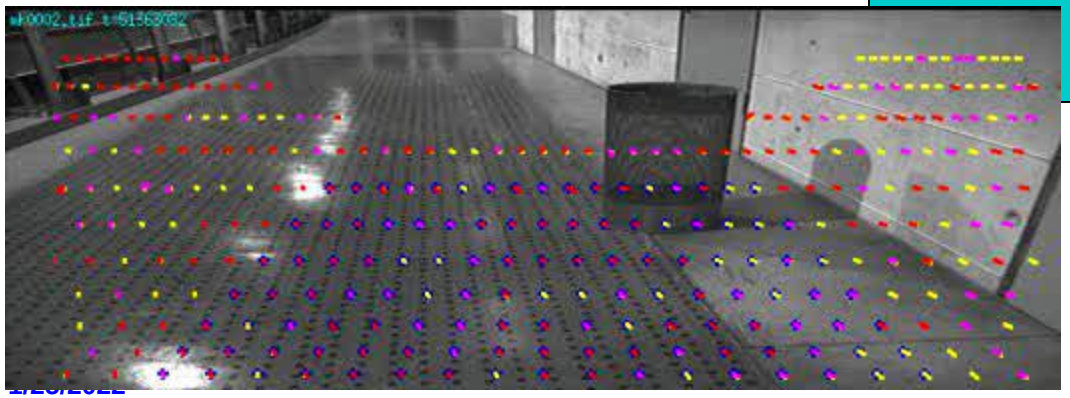
- Draw optic flow vector u

Optic flow /Visual motion detecton

- Relating two adjacent frames: (small differences):

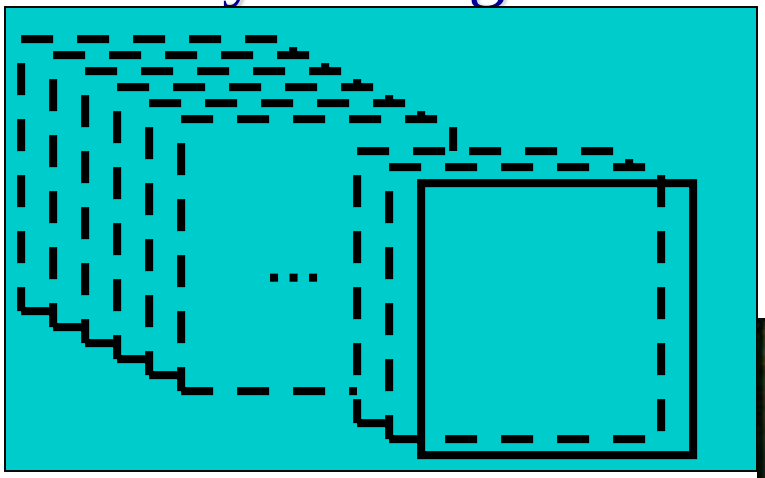
$$\mathrm{Im}(x+\delta x,\ \ y+\delta y,\ \ t+1)\ \cong\ \mathrm{Im}(x,\ \ y,\ \ t)$$

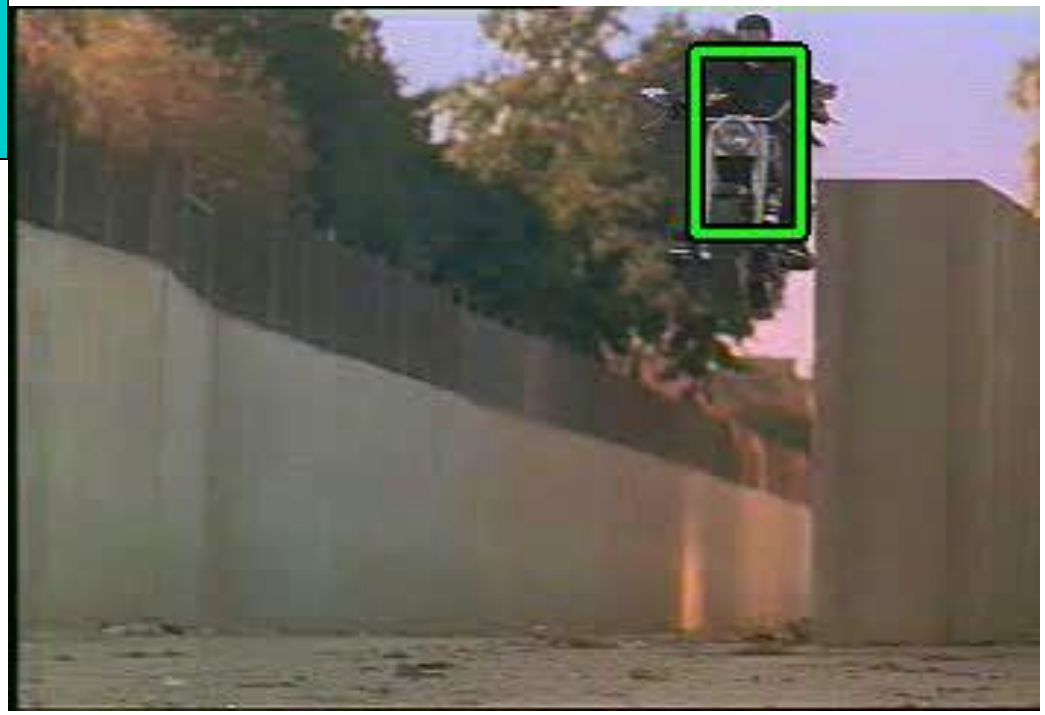




“Visual Tracking” / Stabilization

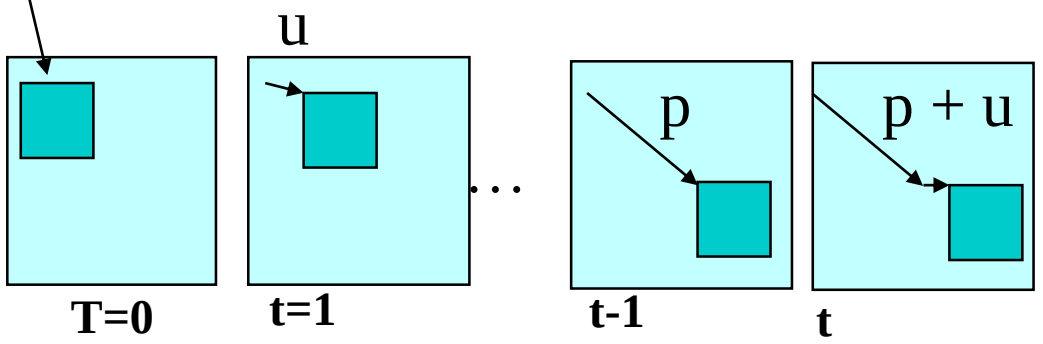
- Globally relating all frames: (large differences):





Manually select template T in first video frame

- Manually selected template T = I(t=0)
- Matlab ginput()



Optic flow

- Compute u between frames

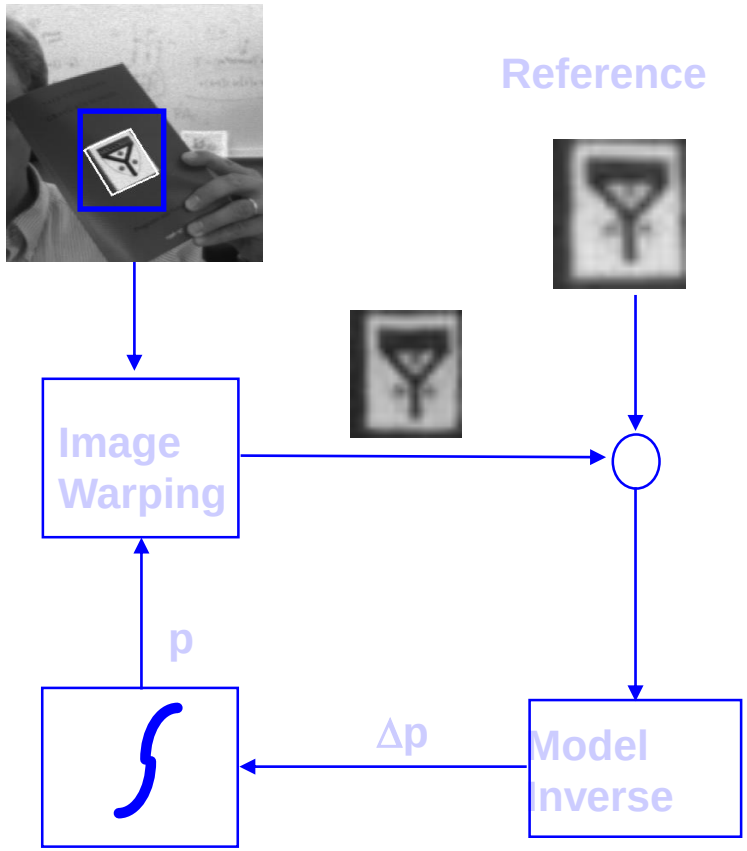
Tracking:

- Keep track of global motion **p**

MTF Usage Example – Multi Target Tracking

Tracking Cycle

- Prediction
  - Prior states predict new appearance
- Image warping
  - Generate a “normalized view”
- Model inverse
  - Compute error from nominal
- State integration
  - Apply correction to state

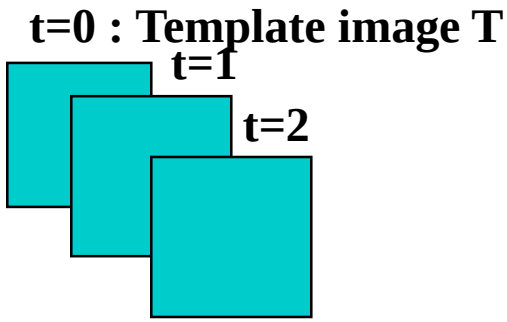


From Optic Flow to Tracking

- Solving for the translational motion of a patch

Over determined equation system:

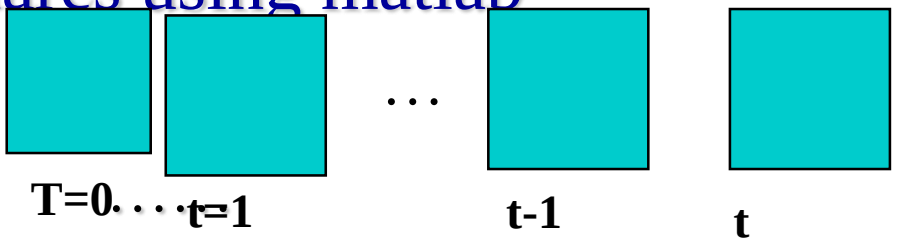
$$\begin{pmatrix} \vdots \\ -\frac{\partial \mathrm{Im}}{\partial t} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \vdots \\ \frac{\partial \mathrm{Im}}{\partial x} & \frac{\partial \mathrm{Im}}{\partial y} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix}$$



$$-\mathrm{Im\_t} = \mathrm{M}\,u$$

- Can be solved in least squares using matlab

$$u = \mathrm{M}\backslash \mathrm{Im\_t}$$



- Update tracking state:

$$p = p + u$$

- For t=2,2,… Im_t = Im(t,x+p) – Im(0, x)

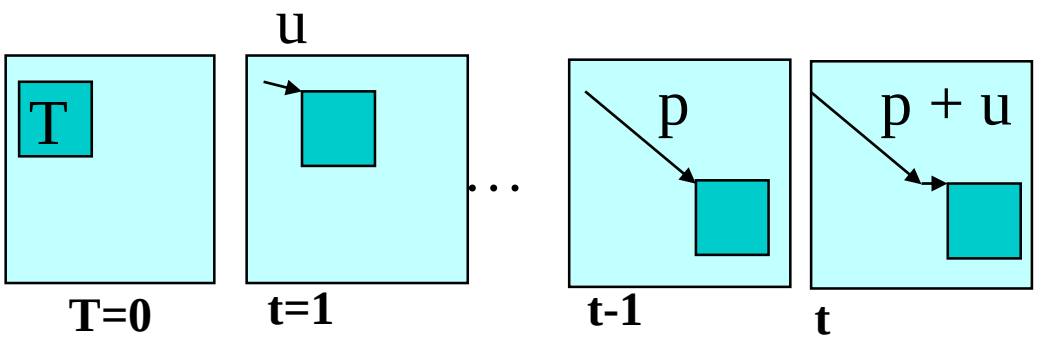
Tracking Lucas-Kanade algorithm

- Create tracking loop, iterate for each new image

Init p=0, Template T

For t = 1…

1. Receive I(t+1)
2. Compute dIm = I(t+1, x+p) – T
3. Solve -Im_t = M u
   - Use u = M\Im_t
4. Update p = p + u



Notice: Template is Manually selected

Tracking Lucas-Kanade algorithm

- Create tracking loop, iterate for each new image

Init p=0, Template T

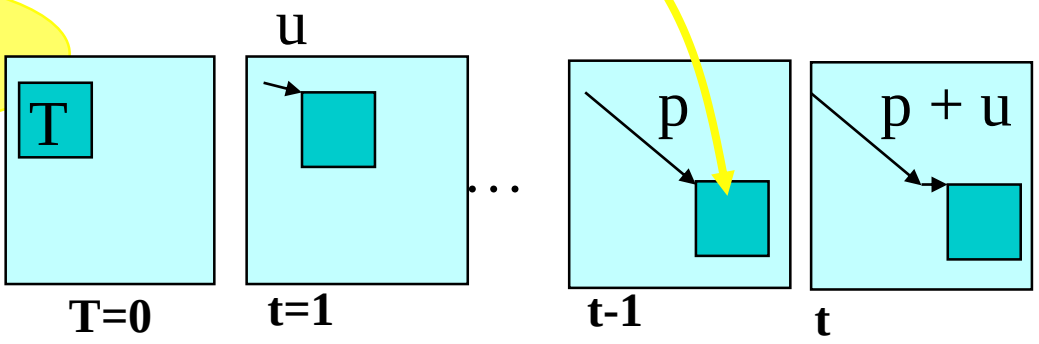
For t = 1…

1. Receive I(t+1)
2. Compute dIm = I(t+1, x+p) – T
3. Solve -Im_t = M u
   - Use u = M\Im_t
4. Update p = p + u

Notice: Template is

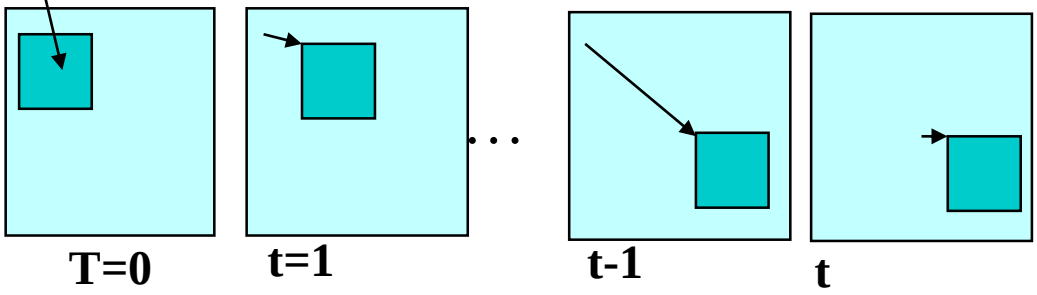
Manually selected

In first frame



Manually select template T in first video frame

- Manually selected:
- Matlab ginput()



MTF Usage Example – Multi Target Tracking

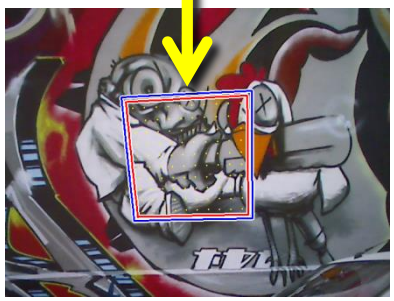
Registration: from trans **u** to warp **w(x,p)**

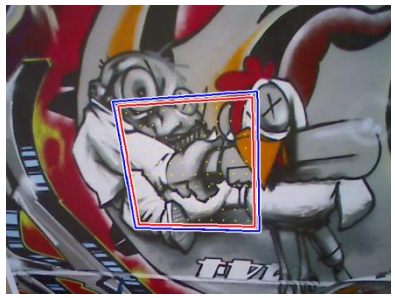
Find parameters of a warping function such that:

$$\mathcal{I}\left(\mathbf{w}\left(\bar{\mathbf{x}};\mathbf{p}_i\right)\right) = \mathcal{I}_T\left(\mathbf{p}_i\right)$$

for all template points $\mathbf{p}_i$

$\mathbf{w}(\bar{\mathbf{x}};\mathbf{p}_i)$





Intensity-based Template Tracking

Goal

Find parameters of a warping function such that:

$$\mathcal{I}\left(\mathbf{w}\left(\bar{\mathbf{x}};\mathbf{p}_i\right)\right) = \mathcal{I}_T\left(\mathbf{p}_i\right)$$

for all template points $\mathbf{p}_i$

Reformulate the goal using a prediction $\hat{\mathbf{x}}$ as approximation of $\bar{\mathbf{x}}$:

- Find the **parameters' increment** $\Delta\mathbf{x}$ such that $\hat{\mathbf{x}} + \Delta\mathbf{x} = \bar{\mathbf{x}}$
- by **minimizing** $\|\mathbf{y}(\Delta\mathbf{x})\| = \left\|\left[y_1(\Delta\mathbf{x})\ y_2(\Delta\mathbf{x})\ \ldots\ y_n(\Delta\mathbf{x})\right]^T\right\|$

$$y_i(\Delta\mathbf{x}) = \mathcal{I}(\mathbf{w}(\hat{\mathbf{x}} + \Delta\mathbf{x};\mathbf{p}_i)) - \mathcal{I}_T(\mathbf{p}_i)$$

This is a **non-linear minimization problem**.

Intensity-based Template Tracking
Lukas-Kanade

Uses the **Gauss-Newton method** for minimization:

- Applies a first-order Taylor series approximation
- Minimizes iteratively

B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision, 1981.

Lukas-Kanade
Approximation by linearization

First-order Taylor series approximation:

$$\mathbf{y}(\Delta\mathbf{x}) \approx \mathbf{y}(\mathbf{0}) + \mathbf{J}_{\mathbf{y}}(\mathbf{0})\Delta\mathbf{x}$$

where the **Jacobian matrix** is:

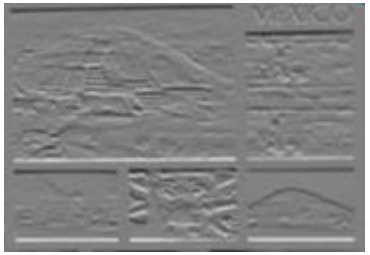
$$\mathbf{J}_{\mathbf{y}}(\mathbf{0}) = \left.\frac{\partial \mathbf{y}(\Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=\mathbf{0}}$$

$$\mathbf{J}_{y_i}(\mathbf{0}) = \left.\frac{\partial \mathcal{I}(\mathbf{p})}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{w}(\hat{\mathbf{x}};\mathbf{p}_i)} \cdot \left.\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p}_i)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}}$$

$$= \mathbf{J}_{\mathcal{I}}(\mathbf{w}(\hat{\mathbf{x}};\mathbf{p}_i)) \cdot \mathbf{J}_{\mathbf{w}(\cdot;\mathbf{p}_i)}(\hat{\mathbf{x}})$$

Jacobian of the **current** image

Jacobian of the



gradient images

Image Derivatives
4DoF

- Generalize to many Degrees of freedooms (DOFs)

$$\delta I = \|I - I_i\|$$

$$\delta I = M\delta\mu$$

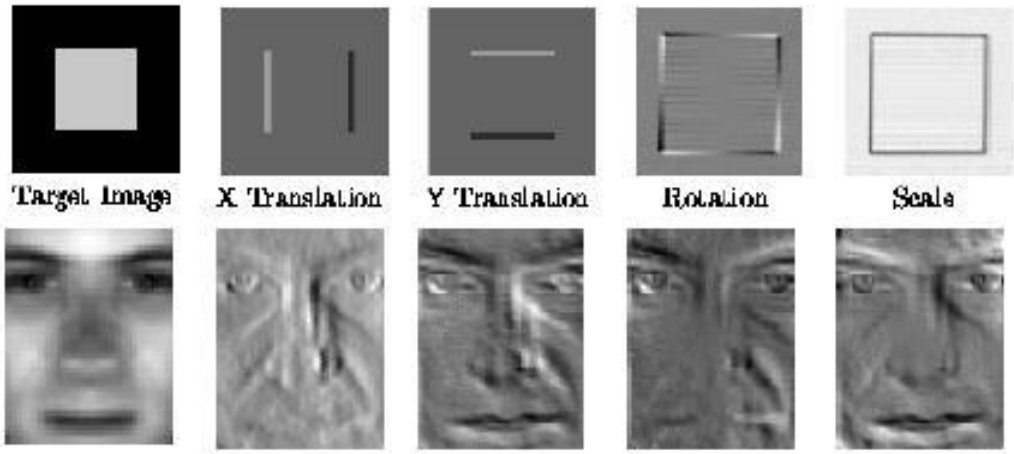
$$M = [I_x|I_y|I_r|I_s]$$

$$I_x = I(x,y) - I(x-1,y)$$

$$I_y = I(x,y) - I(x,y-1)$$

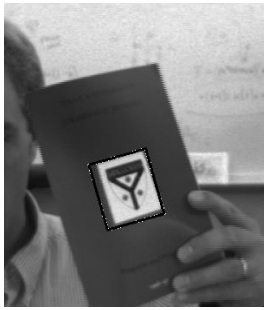
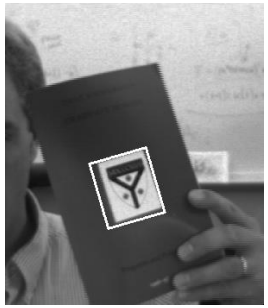
$$I_r = -yI_x + xI_y$$

$$I_s = \frac{1}{\sqrt{x^2+y^2}}(xI_x + yI_y)$$



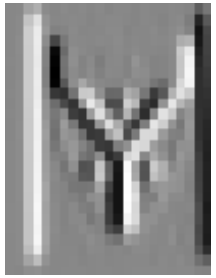
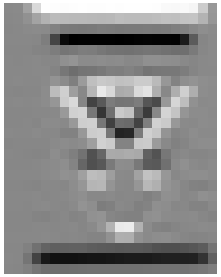
Im = Mu

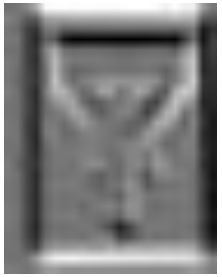
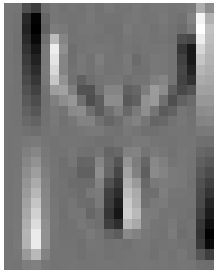
Image Derivatives
6 DoF


Template

Difference images $\mathrm{M(u)} = \partial \mathrm{Im} / \partial \mathrm{u}$

X **Y** **Rotation** **Scale** **Aspect** **Shear**

Planar Texture Variability 1
Affine Variability

- Affine warp function

$$\begin{bmatrix} u_w \\ v_w \end{bmatrix} = \mathcal{W}_a(\mathbf{p}, \mathbf{a}) = \begin{bmatrix} \mathbf{a_3} & \mathbf{a_4} \\ \mathbf{a_5} & \mathbf{a_6} \end{bmatrix} \mathbf{p} + \begin{bmatrix} \mathbf{a_1} \\ \mathbf{a_2} \end{bmatrix}$$

- Corresponding image variability

$$\Delta \mathbf{I}_a = \sum_{i=1}^{6} \frac{\partial}{\partial a_i} \mathbf{I}_w \Delta a_i = \left[\frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}\right] \begin{bmatrix} \frac{\partial u}{\partial a_1} & \cdots & \frac{\partial u}{\partial a_6} \\ \frac{\partial v}{\partial a_1} & \cdots & \frac{\partial v}{\partial a_6} \end{bmatrix} \begin{bmatrix} \Delta a_1 \\ \vdots \\ \Delta a_6 \end{bmatrix}$$

- Discretized for images

$$\Delta \mathbf{I_a} = \left[\frac{\partial \mathbf{I}}{\partial \mathbf{u}}, \frac{\partial \mathbf{I}}{\partial \mathbf{v}}\right] \begin{bmatrix} \mathbf{1} & \mathbf{0} & * \mathbf{u} & \mathbf{0} & * \mathbf{v} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & * \mathbf{u} & \mathbf{0} & * \mathbf{v} \end{bmatrix} \begin{bmatrix} \mathbf{y_1} \\ \vdots \\ \mathbf{y_6} \end{bmatrix}$$

$$= [\mathbf{B}_1 \ldots \mathbf{B}_6][y_1, \ldots, y_6]^T = B_a \mathbf{y_a}$$

Planar Texture Variability 2
Projective Variability

- Homography warp

$$\begin{bmatrix} u' \\ v' \\ \end{bmatrix} = \mathcal{W}_h(\mathbf{x}_h, \mathbf{h}) = \frac{1}{1+h_7u+h_8v} \begin{bmatrix} h_1u & h_3v & h_5 \\ h_2u & h_4v & h_6 \end{bmatrix}$$

- Projective variability:

$$\Delta \mathbf{I}_h = \frac{1}{c_1}\left[\frac{\partial \mathbf{I}}{\partial u}, \frac{\partial \mathbf{I}}{\partial v}\right] \begin{bmatrix} u & 0 & v & 0 & 1 & 0 & -\frac{uc_2}{c_1} & -\frac{vc_2}{c_1} \\ 0 & u & 0 & v & 0 & 1 & -\frac{uc_3}{c_1} & -\frac{vc_3}{c_1} \end{bmatrix} \begin{bmatrix} \Delta h_1 \\ \vdots \\ \Delta h_8 \end{bmatrix}$$

$$= [\mathbf{B}_1 \ldots \mathbf{B}_8][y_1, \ldots, y_8]^T = B_h \mathbf{y}_h$$

- Where $c_1 = 1 + h_7u + h_8v$, $c_2 = h_1u + h_3v + h_5$

  and $c_3 = h_2u + h_4v + h_6$

Lukas-Kanade
Iterative minimization

The following steps will be **iteratively** applied:

- **Minimize** a sum of squared differences

$$f(\Delta \mathbf{x}) = \frac{1}{2} \parallel \mathbf{y}\,(\mathbf{0}) + \mathbf{J}_{\mathbf{y}}(\mathbf{0})\Delta \mathbf{x} \parallel^2$$

where the parameter increment has a **closed form solution** (Gauss-Newton)

$$\Delta \mathbf{x} = -\mathbf{H}^{-1}\mathbf{J}_{\mathbf{y}}(\mathbf{0})^T\mathbf{y}\,(\mathbf{0})$$

$$\mathbf{H} = \mathbf{J}_{\mathbf{y}}(\mathbf{0})^T\mathbf{J}_{\mathbf{y}}(\mathbf{0})$$

  - **Better (Matlab): dx = J\y0**

- **Update** the parameter approximation

$$\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \Delta \mathbf{x}$$

This is repeated **until convergence** is reached.

Compare simple case translation

- Create tracking loop, iterate for each new image

Init p=0, Template T

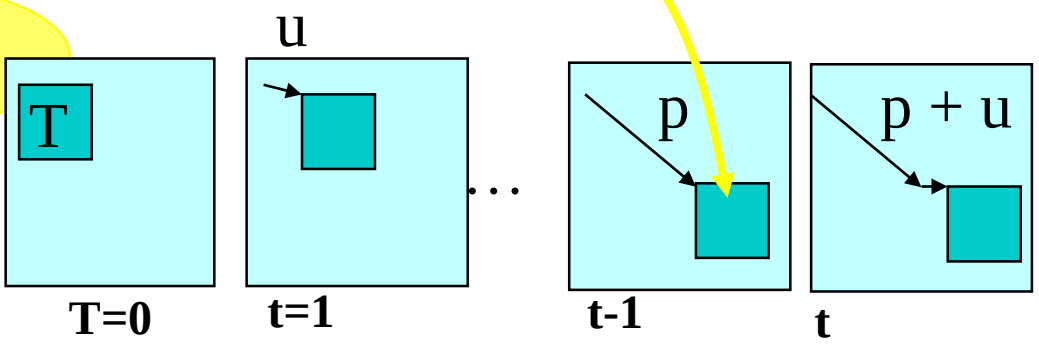
For t = 1…

1. Receive I(t+1)
2. Compute dIm = I(t+1, x+p) – T
3. Solve -Im_t = M u
   - Use u = M\Im_t
4. Update p = p + u

Notice: Template is

Manually selected

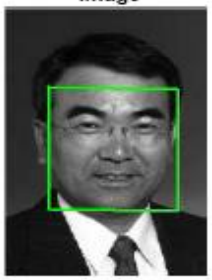
In first frame



Template

$T(\mathbf{x})$

Image



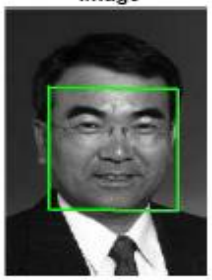
$I(\mathbf{x})$

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

Template

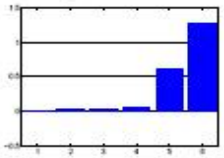
$T(\mathbf{x})$

Image



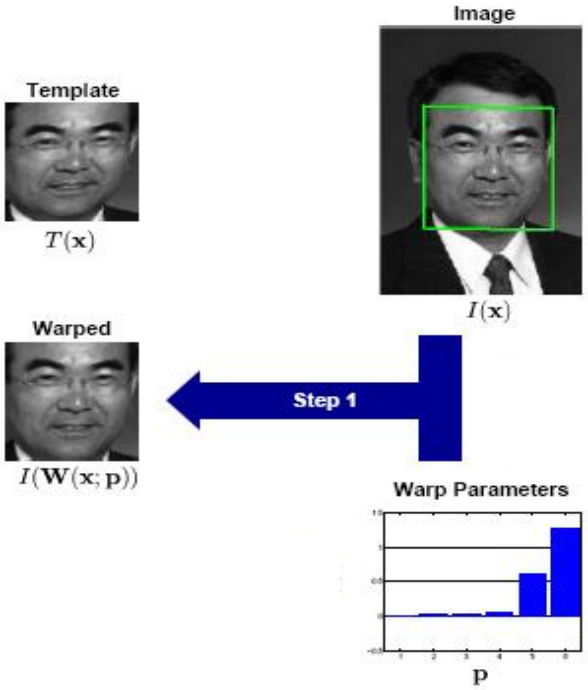
$I(\mathbf{x})$

Warp Parameters

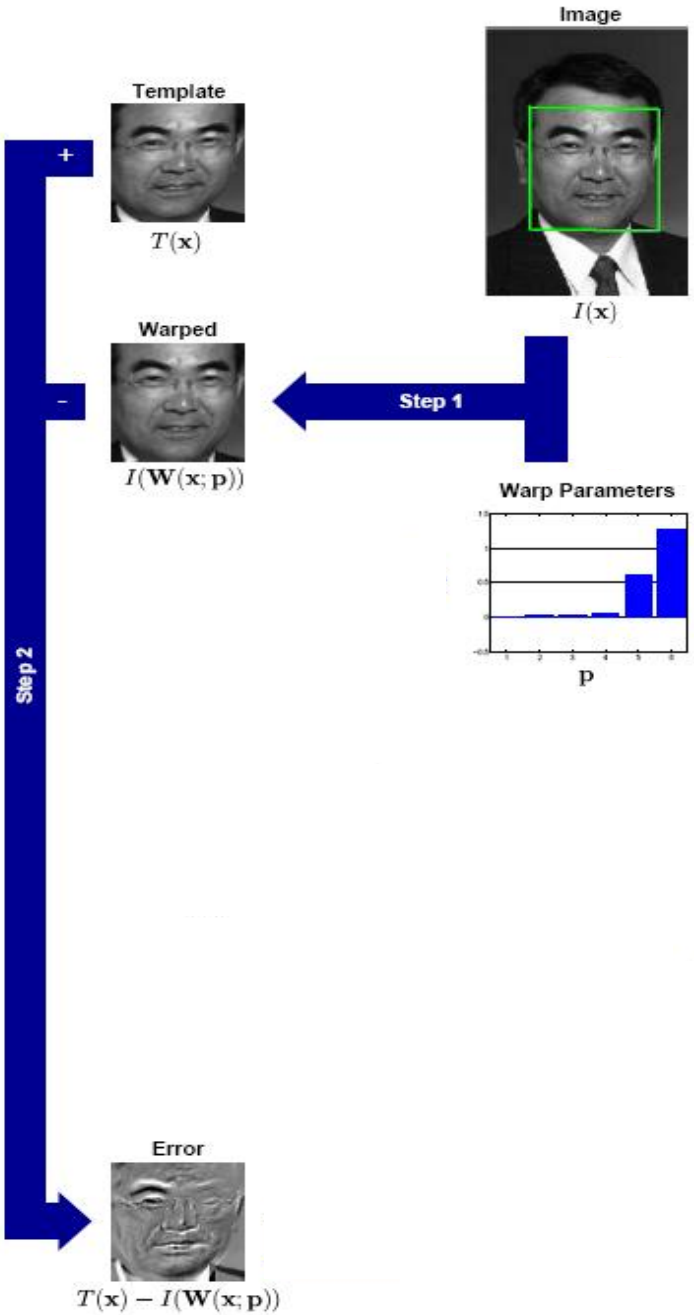


$\mathbf{p}$

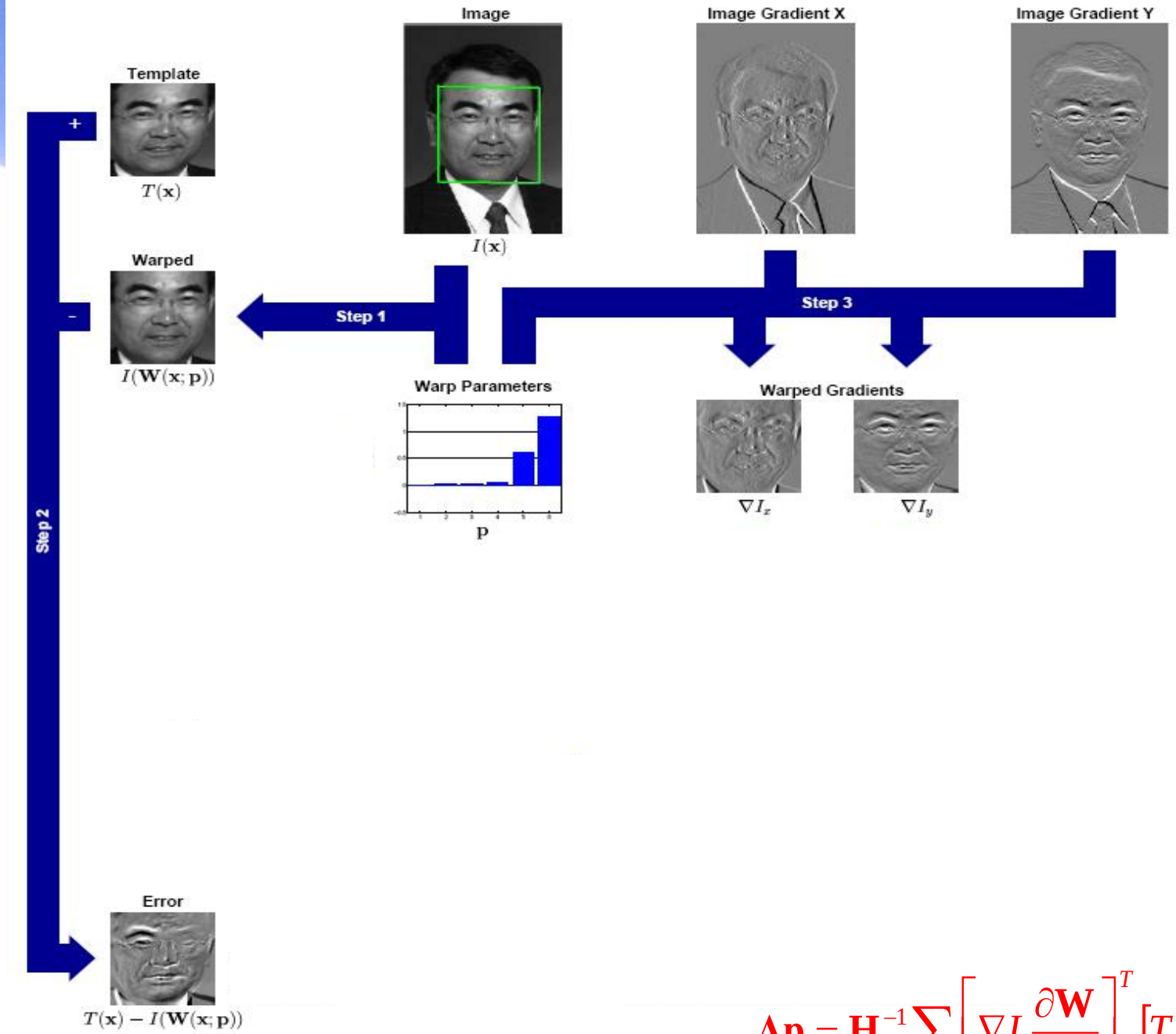
$$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$$



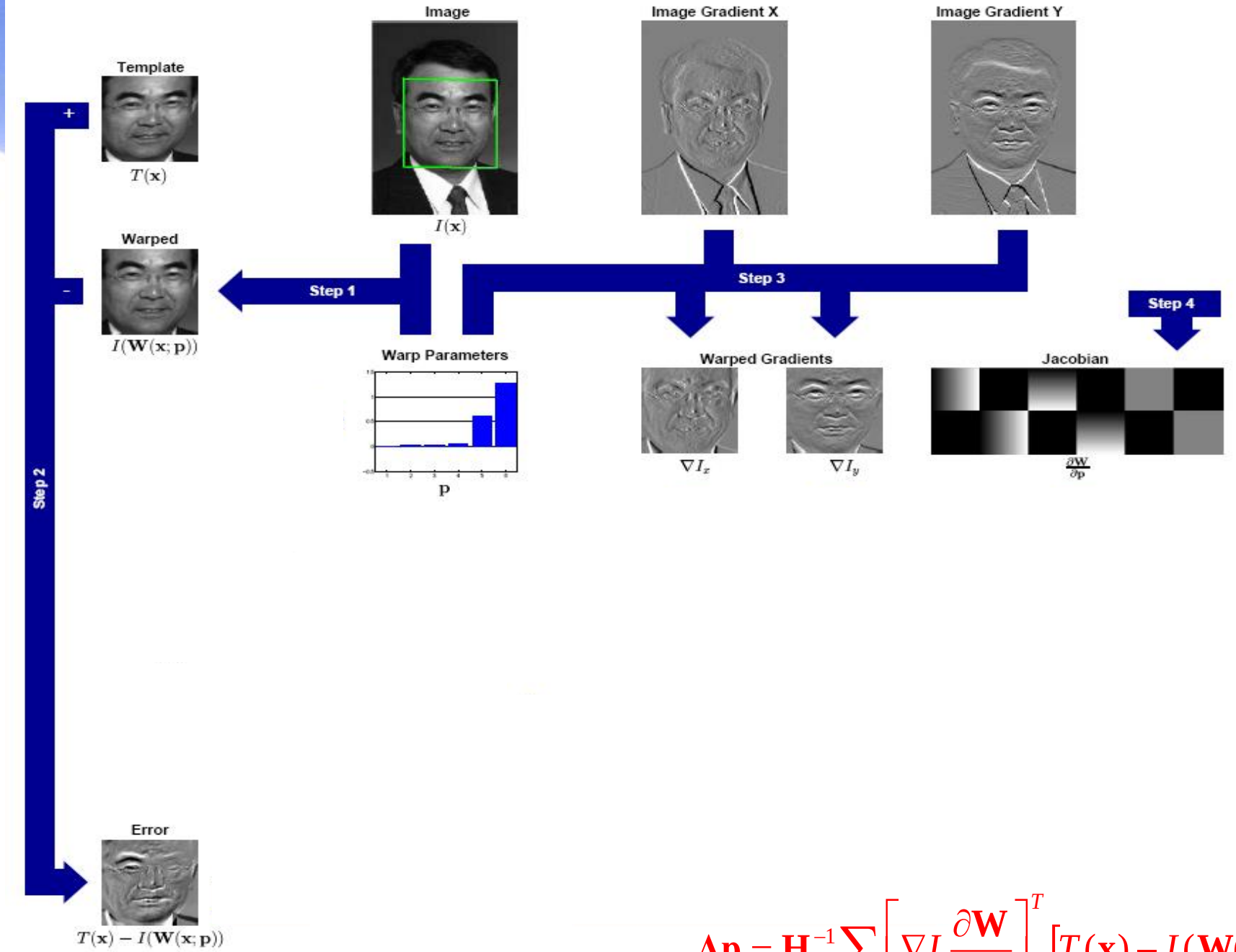
$$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$$



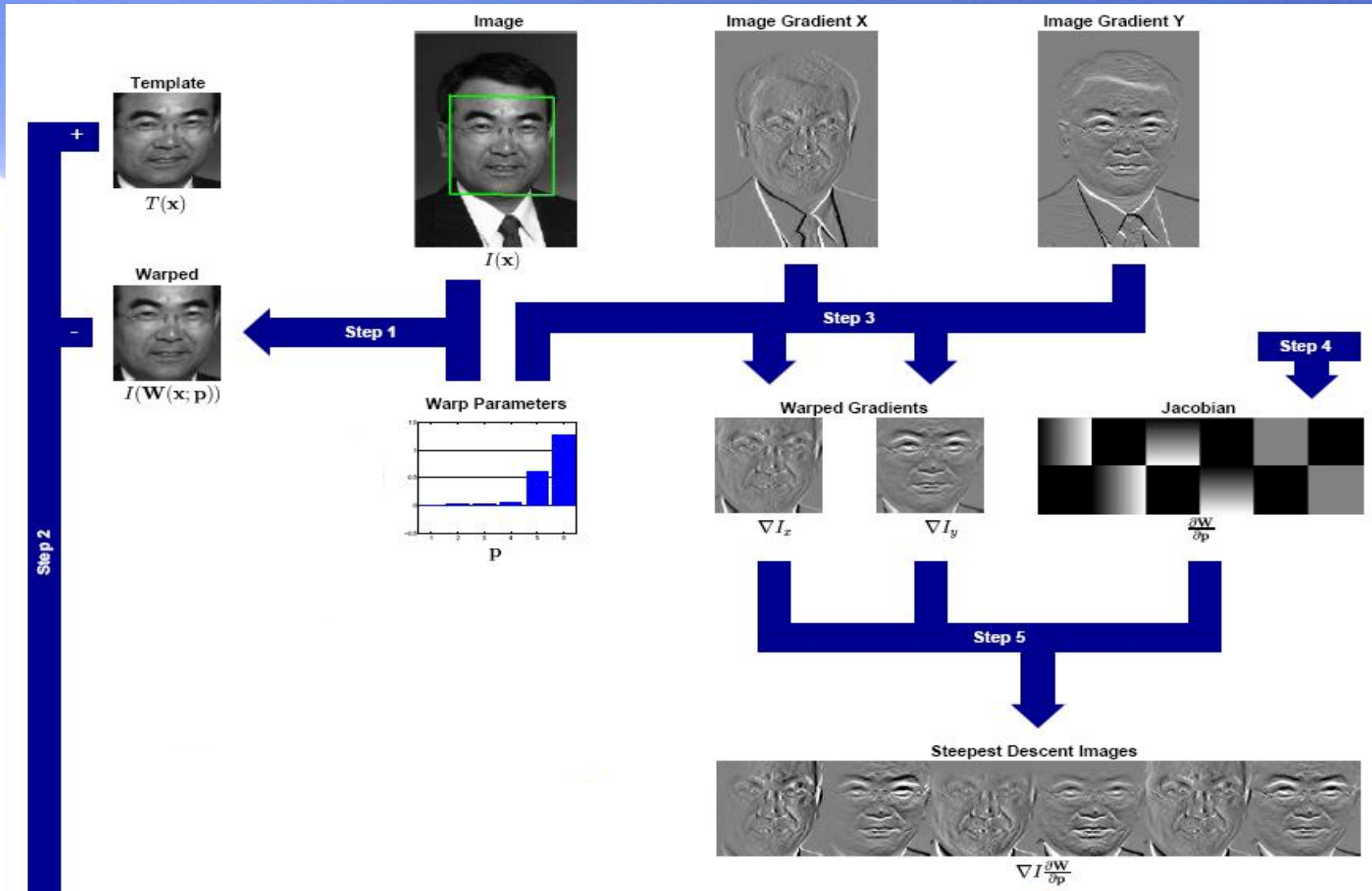
$$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{T} \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$$



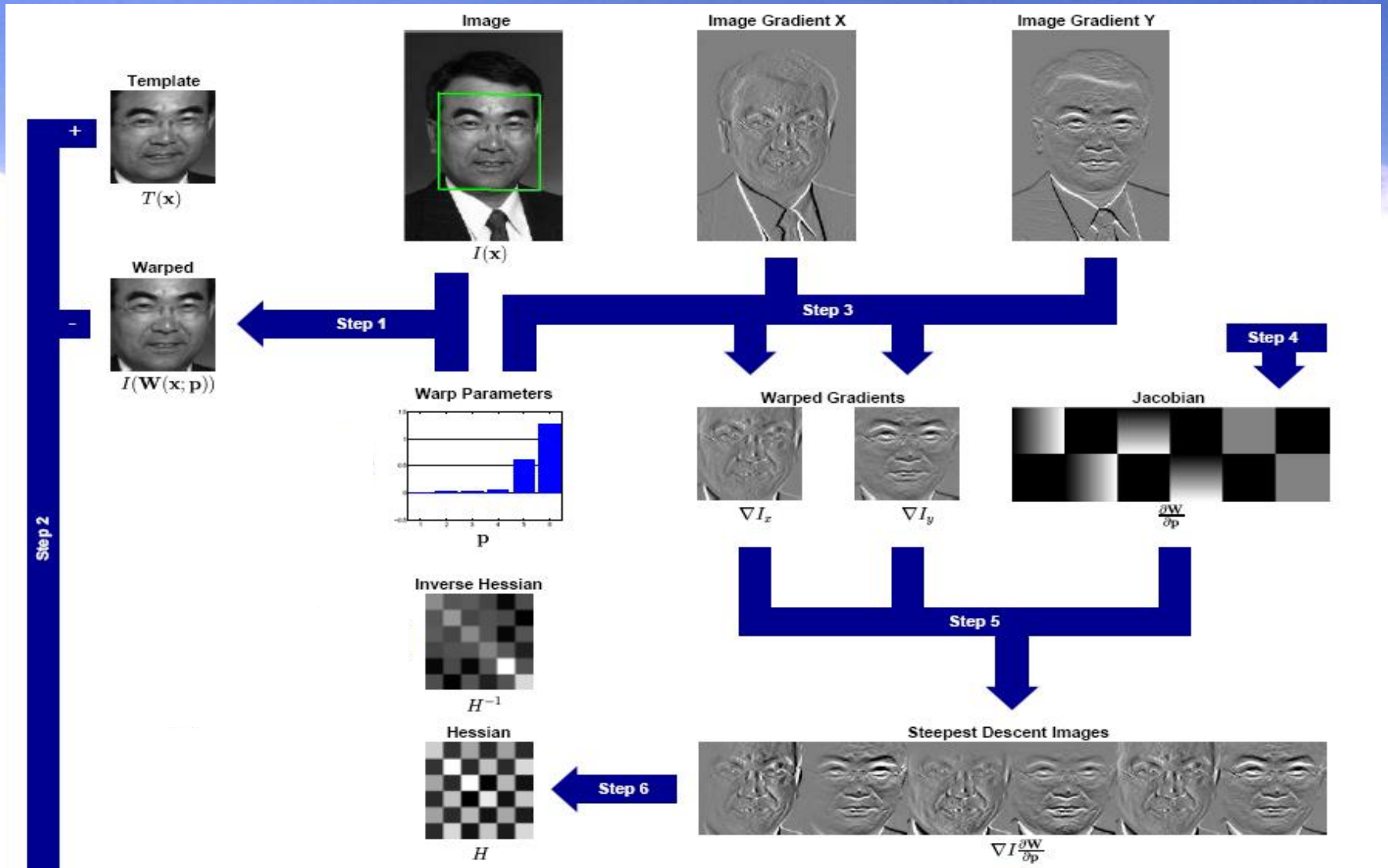
$$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$$



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



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



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

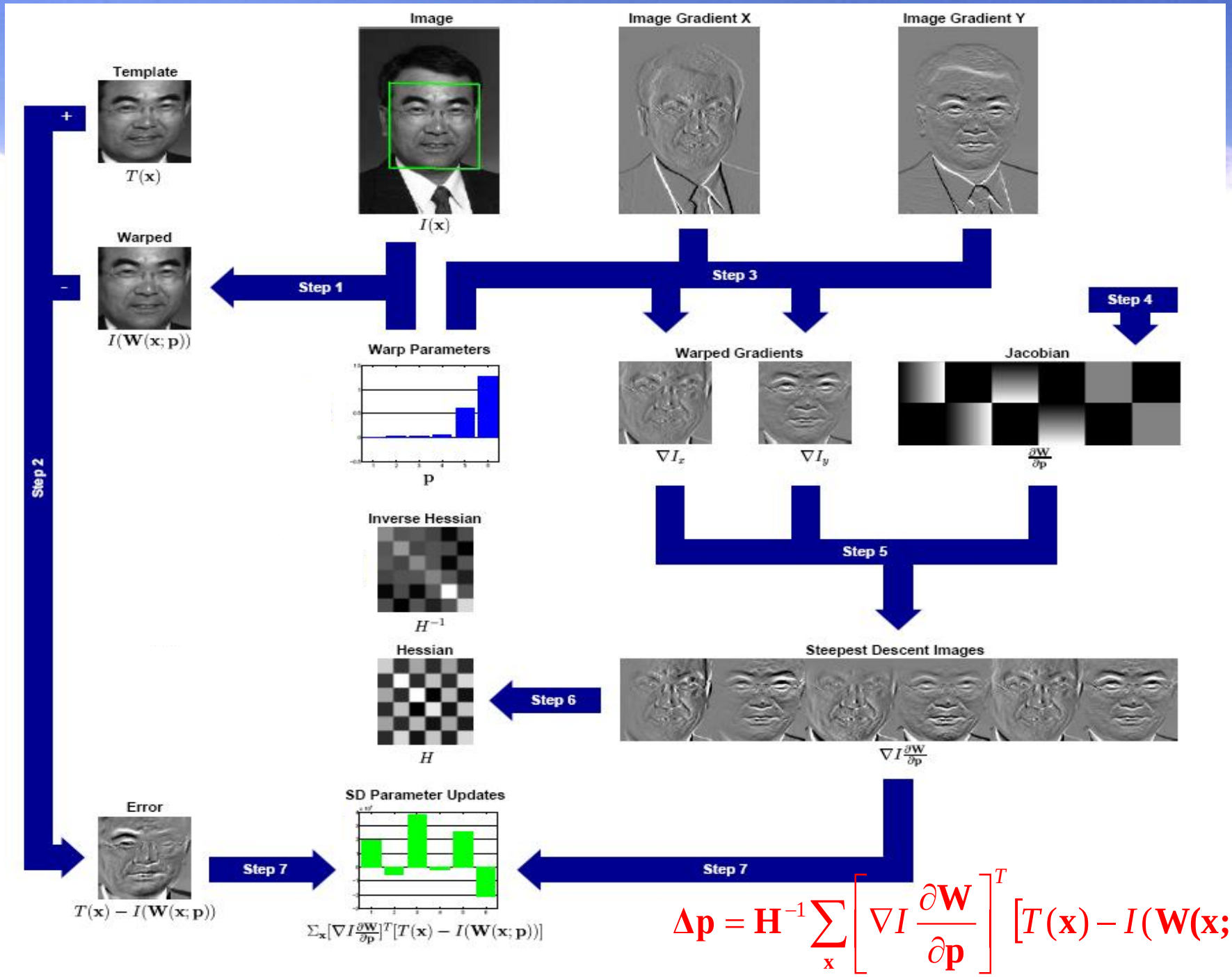
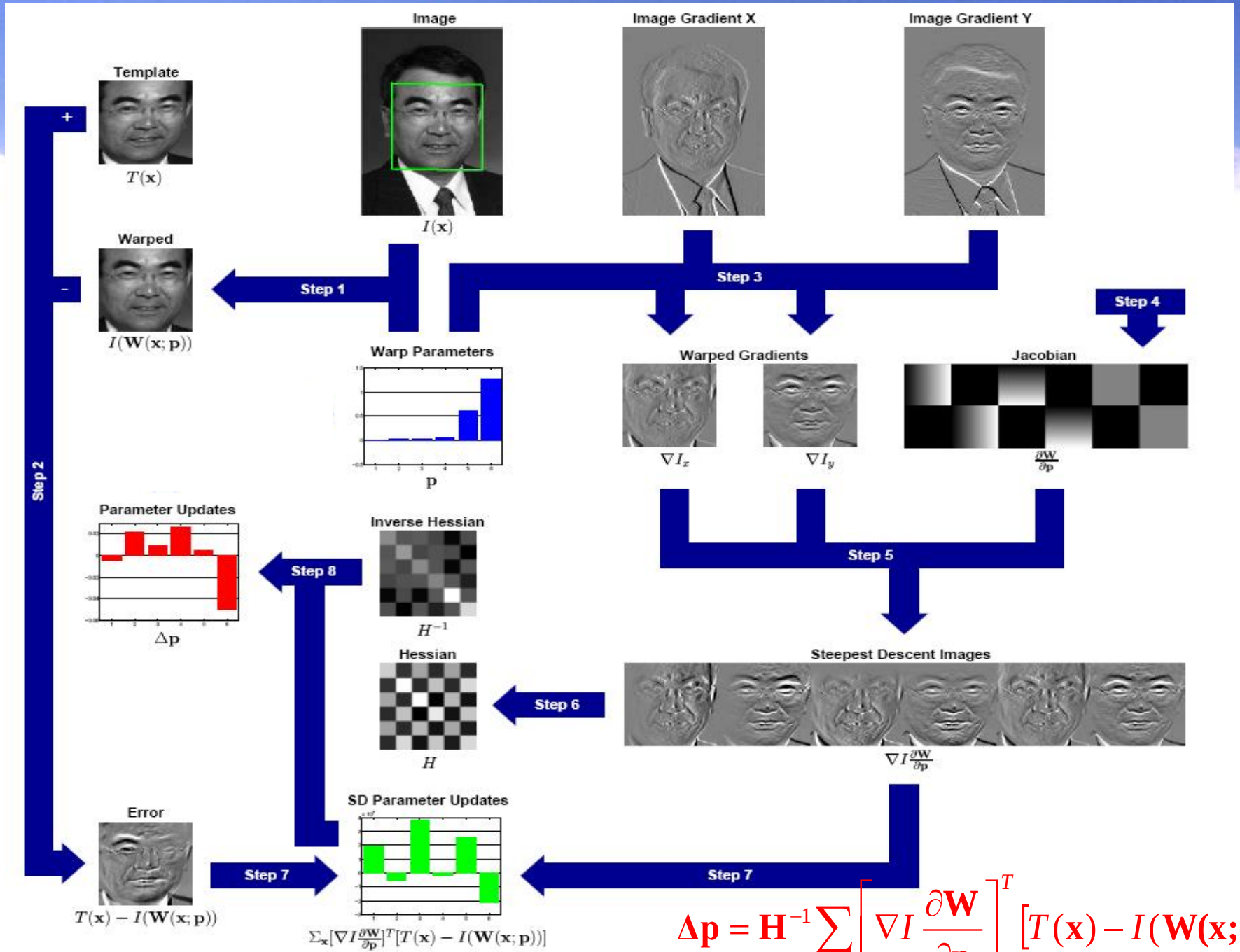
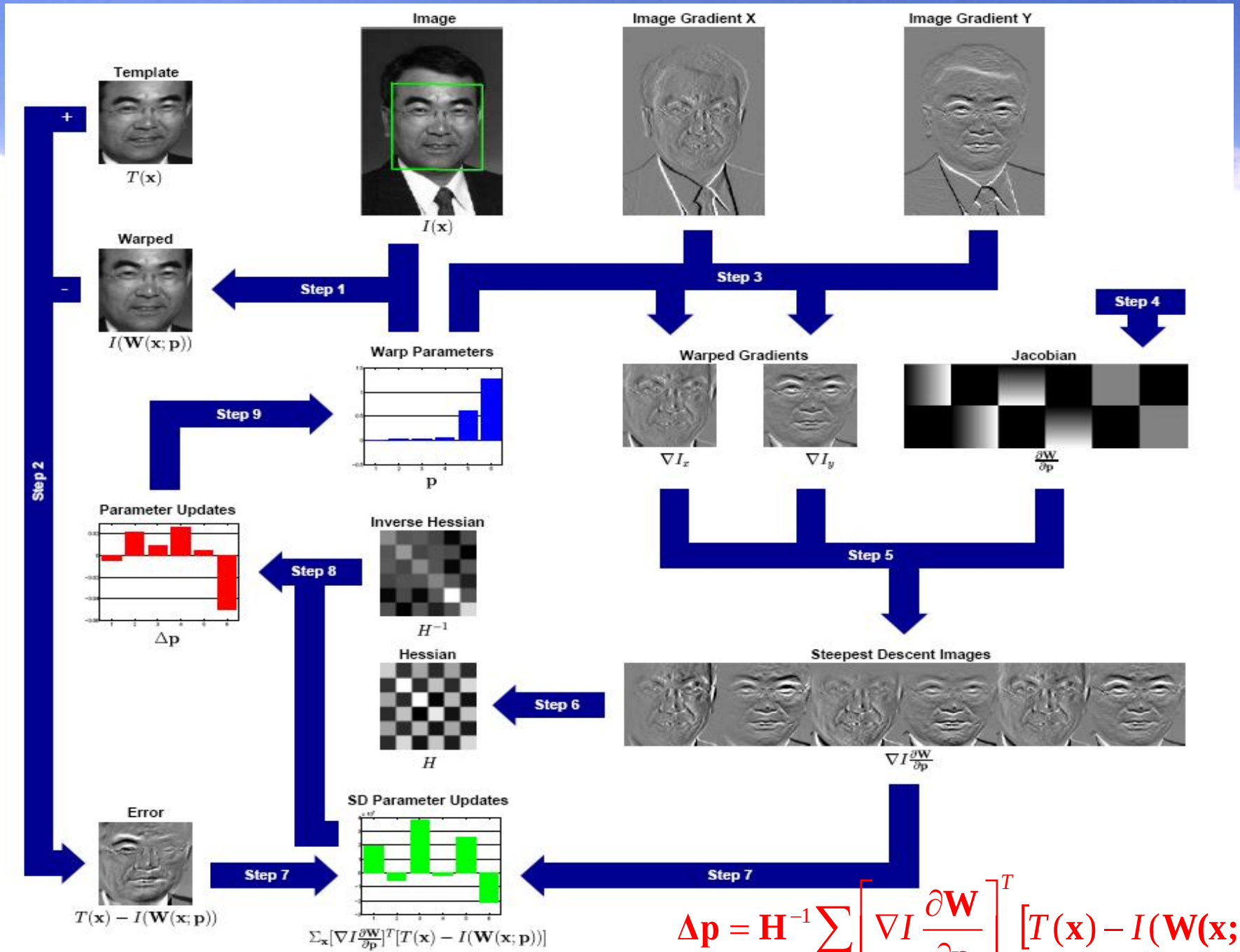


Image
Image Gradient X
Image Gradient Y
Template
$T(\mathbf{x})$
$I(\mathbf{x})$
Warped
$I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
+
-
Step 1
Step 2
Step 3
Step 4
Warp Parameters
$\mathbf{p}$
Warped Gradients
∇I_x
∇I_y
Jacobian
$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
Step 5
Inverse Hessian
H^{-1}
Hessian
H
Step 6
Steepest Descent Images
$\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
Error
$T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
Step 7
SD Parameter Updates
$\Sigma_{\mathbf{x}}[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
Step 7
$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$



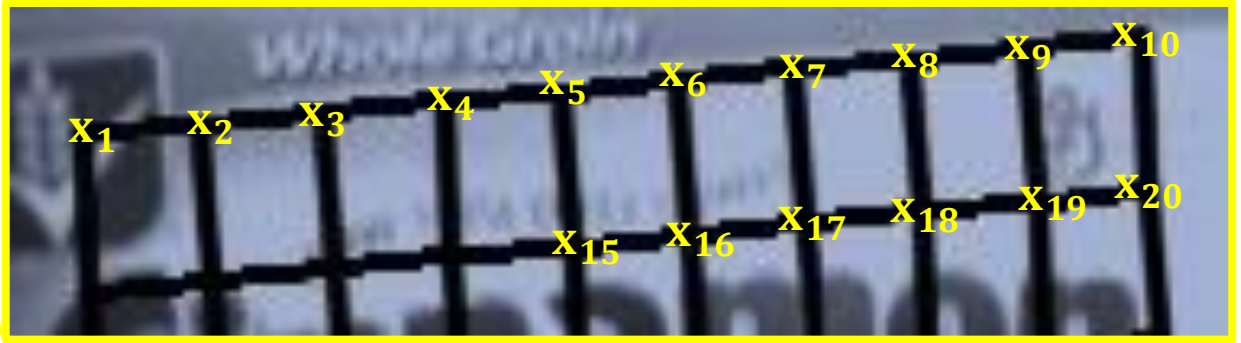
$$\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]$$



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

Registration based Tracking

$$\mathbf{p_t} = \underset{\mathbf{p}}{\operatorname{argmax}}\ f(\mathbf{I_0(x)}, \mathbf{I_t(w(x,p))})$$



Inverse Compositional
Overview

Differences to the Lukas-Kanade algorithm

– Reformulation of the goal:

$$y_i(\Delta\mathbf{x}) = \mathcal{I}_T\left(\mathbf{w}\left(\Delta\mathbf{x};\mathbf{p}_i\right)\right) - \mathcal{I}\left(\mathbf{w}\left(\hat{\mathbf{x}};\mathbf{p}_i\right)\right)$$

$$\begin{aligned}
\mathbf{J}_{y_i}(\mathbf{0}) &= \left.\frac{\partial y_i(\Delta\mathbf{x})}{\partial\Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=\mathbf{0}} \\
&= \left.\frac{\partial\mathcal{I}_T\left(\mathbf{w}\left(\Delta\mathbf{x};\mathbf{p}_i\right)\right)}{\partial\mathbf{w}\left(\Delta\mathbf{x};\mathbf{p}_i\right)} \cdot \frac{\partial\mathbf{w}\left(\Delta\mathbf{x};\mathbf{p}_i\right)}{\partial\Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=\mathbf{0}} \\
&= \left.\frac{\partial\mathcal{I}_T\left(\mathbf{p}\right)}{\partial\mathbf{p}}\right|_{\mathbf{p}=\mathbf{w}(\mathbf{0};\mathbf{p}_i)=\mathbf{p}_i} \cdot \left.\frac{\partial\mathbf{w}\left(\Delta\mathbf{x};\mathbf{p}_i\right)}{\partial\Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=\mathbf{0}} \\
&= \mathbf{J}_{\mathcal{I}_T}(\mathbf{p}_i) \cdot \mathbf{J}_{\mathbf{w}(\cdot;\mathbf{p}_i)}(\mathbf{0})
\end{aligned}$$

Jacobian of the **template image**

Jacobian of the **warp**

Inverse Compositional
Overview

Differences to the Lukas-Kanade algorithm

– Reformulation of the goal

$$y_i(\Delta \mathbf{x}) = \mathcal{I}_T\left(\mathbf{w}\left(\Delta \mathbf{x}; \mathbf{p}_i\right)\right) - \mathcal{I}\left(\mathbf{w}\left(\hat{\mathbf{x}}; \mathbf{p}_i\right)\right)$$

→ $\mathbf{J_y}(\mathbf{0})$ and $\mathbf{H} = \mathbf{J_y}(\mathbf{0})^T \mathbf{J_y}(\mathbf{0})$ can be **precomputed**

–

→ only $\mathbf{y}\,(\mathbf{0})$ needs to be computed at each iteration

– Parameter update changes

$$\mathbf{w}\left(\hat{\mathbf{x}}; \mathbf{p}\right) \leftarrow \mathbf{w}\left(\hat{\mathbf{x}}; \mathbf{p}\right) \circ \mathbf{w}\left(\Delta \mathbf{x}; \mathbf{p}\right)^{-1}$$

S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms, 2001.

Efficient Second-Order Minimization
Short Overview

- Uses **second-order Taylor approximation** of the cost function
  - → **Fewer iterations** needed to converge
  - → **Larger area of convergence**
  - → **Avoiding local minima** close to the global
- Jacobian needs to be computed at each iteration

S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization, 2004.

Lukas-Kanade
Variations

- **Inverse Compositional (IC):**
  reduce time per iteration
- **Efficient Second-Order Minimization (ESM):**
  improve convergence
- **Approach of Jurie & Dhome (JD):**
  reduce time per iteration
  and improve convergence
- **Adaptive Linear Predictors (ALPs):**
  learn and adapt template online

Jurie & Dhome
Overview

- **Motivation:**
  - Computing the Jacobian in every iteration is expensive
  - Good convergence properties are desired
- **Approach of JD:**
  pre-learn relation between image differences and parameter update:

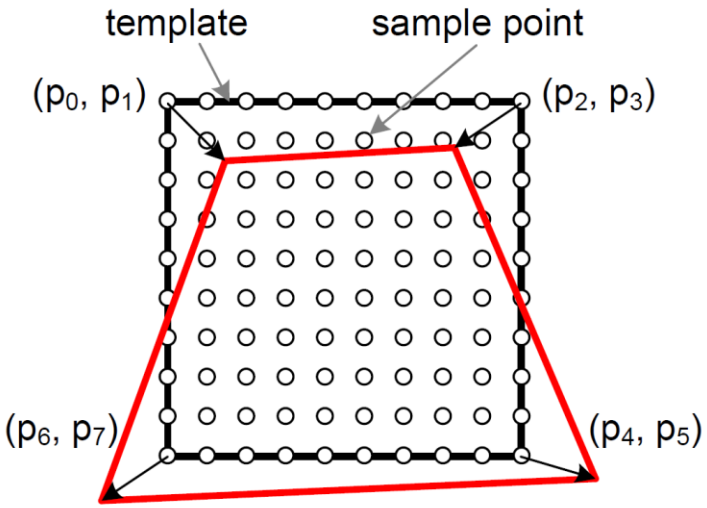
$$\Delta \mathbf{x} = \mathbf{A}\mathbf{y}(\mathbf{0})$$

  - relation $\mathbf{A}$ can be seen as linear predictor
  - $\mathbf{A}$ is fixed for all iterations
  - learning enables to jump over local minima

F. Jurie and M. Dhome. Hyperplane approximation for template matching. 2002

Jurie & Dhome
Template and Parameter Description

- Template consists of n_p sample points
  - Distributed over the template region
  - Used to sample the image data



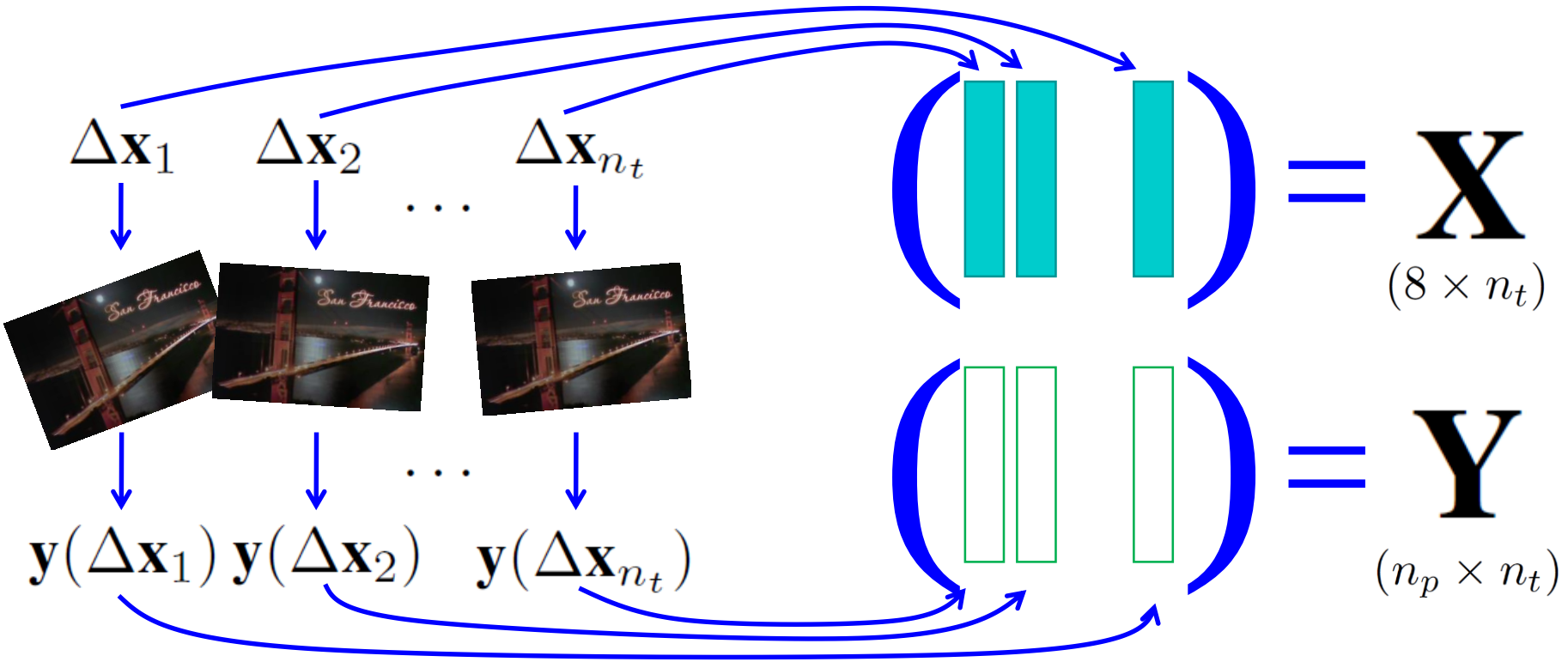
- Deformation is described using the 4 corner points of the template

- Image values are normalized (zero mean, unit standard deviation)

- Error vector $\mathbf{y}(\mathbf{0})$ specifies the differences between the normalized image values

Jurie & Dhome
Learning phase

- Apply set of n_t random transformations to initial template
- Compute corresponding image differences
- Stack together the training data



Jurie & Dhome
Learning phase

- Apply set of n_t random transformations to initial template
- Compute corresponding image differences
- Stack together the training data
- The linear predictor should relate these matrices by:

$$\mathbf{X} = \mathbf{A}\mathbf{Y}$$

- So, the linear predictor can be learned / computed as

$$\mathbf{A} = \mathbf{X}\mathbf{Y}^T \left(\mathbf{Y}\mathbf{Y}^T\right)^{-1}$$

- Note: Matlab: At = Y‘\X‘

Jurie & Dhome
Tracking phase

- **Warp sample points** according to current parameters
- Use warped sample points to extract image values and to **compute error vector** $\mathbf{y}(\mathbf{0})$
- **Compute update** using $\Delta\mathbf{x} = \mathbf{A}\mathbf{y}(\mathbf{0})$
- **Update** current parameters
- **Improve tracking accuracy:**
  - Apply multiple iterations
  - Use multiple predictors trained for different amouts of motions

Jurie & Dhome
Limitations

- Learning large templates is expensive
  → not possible online
- Shape of template cannot be adapted
  → template has to be relearned after each modification
- Tracking accuracy is inferior to LK, IC, …
  → use JD as initialization for one of those

Registration tracking
MTF examples

MTF Usage Example – Multi Target Tracking

UAV Trajectory Estimation

Online Image Mosaicing