

COL7655

Foundations of Visual Computing

Subodh Kumar

subodh@cse.iitd.ac.in

COL7655

Motion Tracking

Thanks to Chetan Arora for help with slides

Track 'Things' in Video

Point, object, feature, template (image)

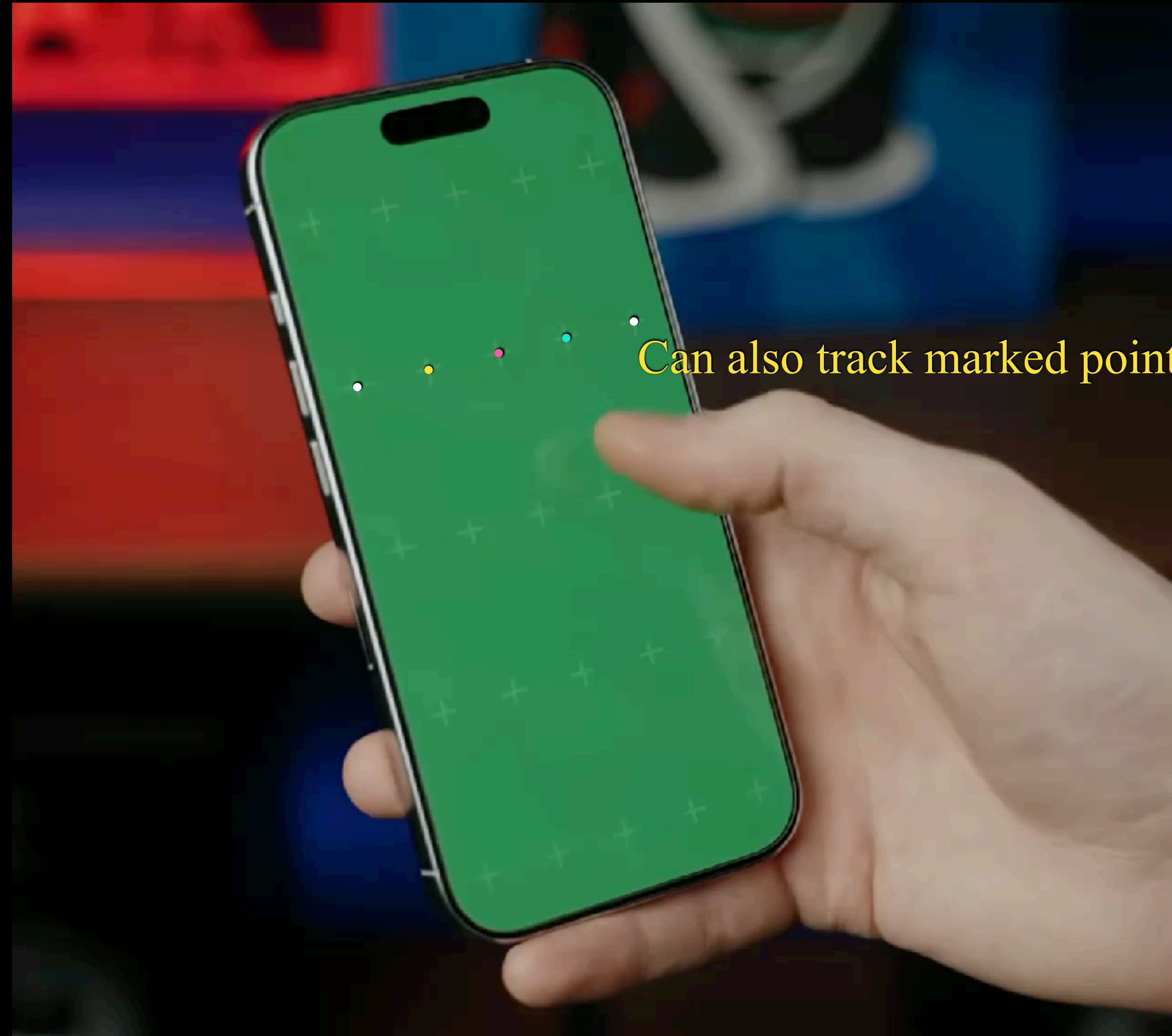
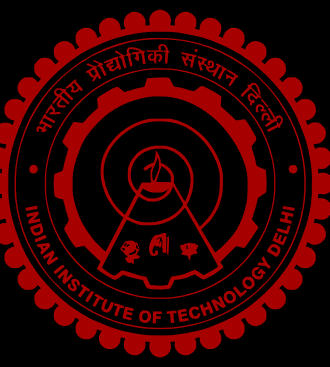


Find a similar thing in the next frame

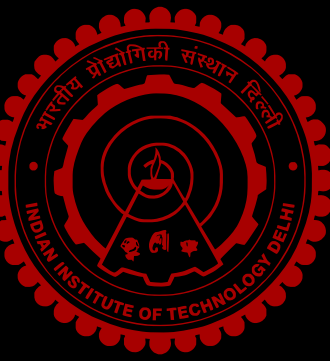
What transformation does it undergo

lighting, occlusion, similar objects

Template Tracking

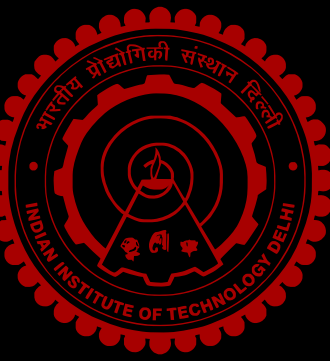


Can also track marked points



Motion Tracking

- Search problem: search for a 'similar object' in the next frame(s)
- Understand the constraints and transformation that objects may undergo
 - Helps narrow down the search
- Challenges:
 - Lighting can change (distance to light, orientation wrt light ..)
 - Object transformation: Shape, orientation (mostly non-rigid due to projection)
 - Occlusion: it could get hidden and reappear, or go out of the frame and come back
 - Both camera and object could move

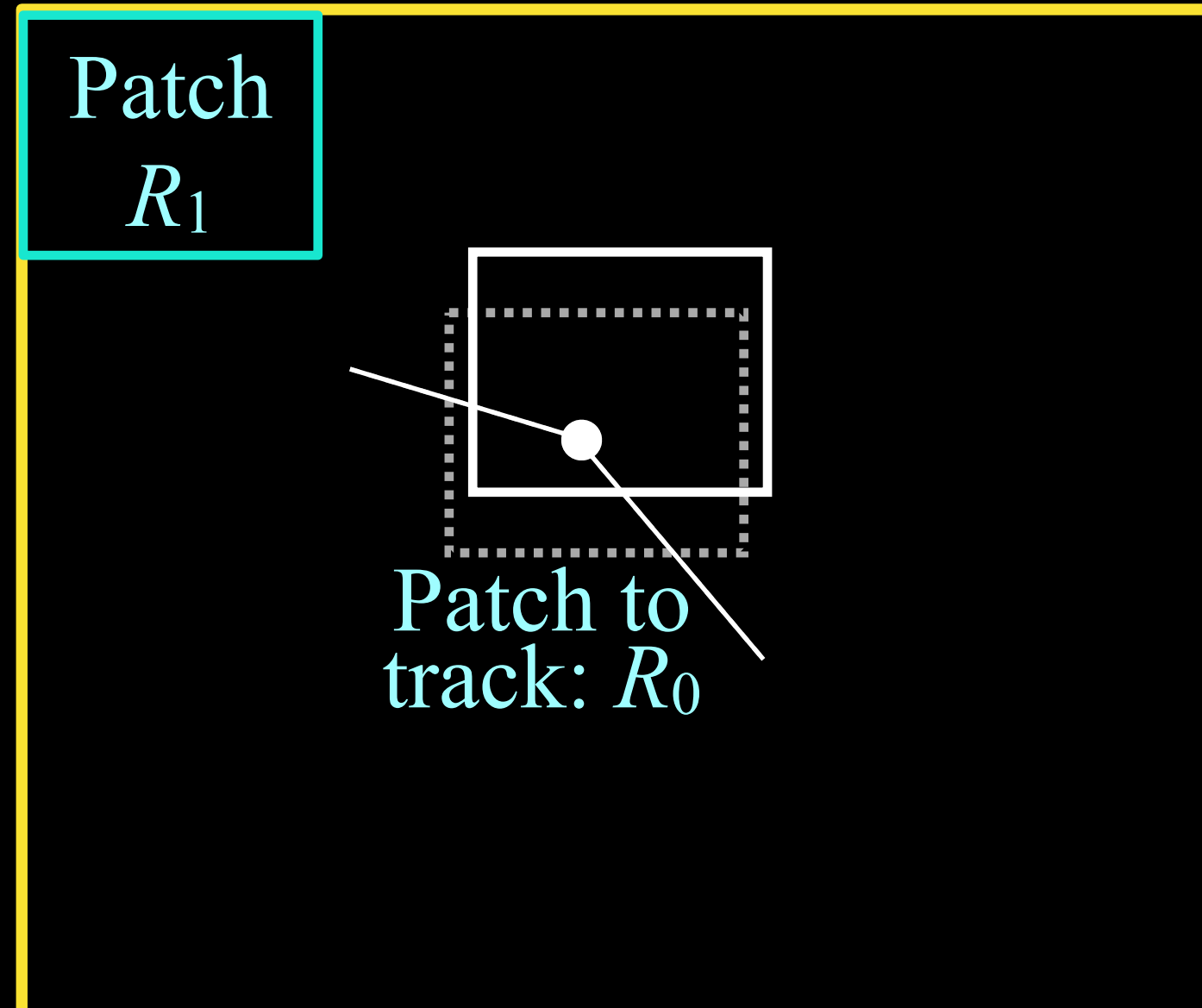


Problem Setup and Basic Approaches

- Positions of object (outline, centroid, bounding region, features...) are given at frame 0
 - Find the positions in frames $i > 0$ (usually from i to $i+1$)
- With fixed camera (*i.e.*, no ‘ego-motion’), constant illumination, and limited object motion, simple background subtraction may suffice:
 - Find overlapping foreground blobs (connected pixels) in two images
- **Block matching**: consider a bounding rectangle R at frame 0, and find the best matching rectangle in frame i , using similarity metrics
- **Differential tracking**: local change at each pixel (no shifted rectangle)
- **Feature matching**: Find features in the next frame that are the closest to the tracked features

Point Tracking with the Block Matching

- Search for rxr patch R_0 of frame I_0 in frame I_1 : compare with all rxr patches of I_1
- Match: Sum of Absolute Difference (SAD), Sum of Squared Distance (SSD), Normalized Cross Correlation (NCC), Zero-normalized Cross Correlation (ZNCC)



$$SAD : \sum_{x,y} |R_1(x, y) - R_0(x, y)| \text{ Not differentiable}$$

$$SSD : \sum_{x,y} (R_1(x, y) - R_0(x, y))^2 \text{ Sensitive to outliers}$$

(aka NCCC: NCC coefficient)

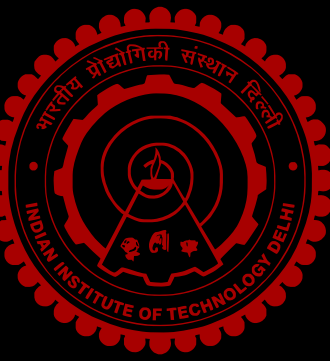
$$ZNCC : \sum_{x,y} (N_1(x, y) * N_0(x, y)), \text{ where}$$

$$N_1(x, y) = \frac{R_1(x, y) - \mu(R_1)}{\sigma(R_1)}$$

$$N_0(x, y) = \frac{R_0(x, y) - \mu(R_0)}{\sigma(R_0)}$$

$$CC : \sum_{x,y} (R_1(x, y) * R_0(x, y)) \text{ Sensitive to brightness}$$

$$NCC : \frac{\sum_{x,y} R_1(x, y) * R_0(x, y)}{\sqrt{\sum_{x,y} R_1^2(x, y)} * \sqrt{\sum_{x,y} R_0^2(x, y)}}$$



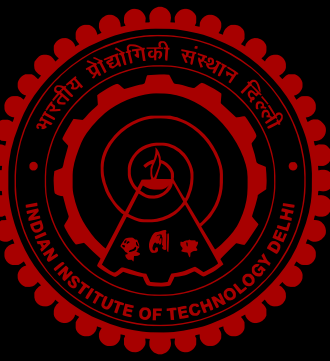
Mean-Shift Tracker

Histogram-based similarity handles some shape deformation

- Start with a box around the object to be tracked
 - Should be relatively distinct from the background
- 1. Find Kernel (e.g., Gaussian) weighted color (e.g., HSV) normalized histogram of box pixels
 - Gives a color $\rightarrow$ probability map
- 2. In the new image, replace each pixel color by the corresponding probability value in the map
 - High value $\Rightarrow$ pixel's color has a high probability of being inside the target box
- 3. Iterate: Stop when the box converges
 - Find the kernel-weighted mean of the probability values within the box
 - Recenter the box at this mean

Mean-Shift Tracker





CAM-Shift Tracker

- Perform mean-shift tracking
- Re-size and re-orient box

Try to align the box to the pixel distribution inside the box

- Expand the search window (for use by the next iteration of mean-shift)



CAM-Shift Tracker

- Perform mean-shift tracking
- Re-size and re-orient box

1. Calculate the 0^{th} , 1^{st} , and 2^{nd} order spatial moments of the converged box

2. New center $x_c, y_c = \frac{M_{01}}{M_{00}}, \frac{M_{10}}{M_{00}}$

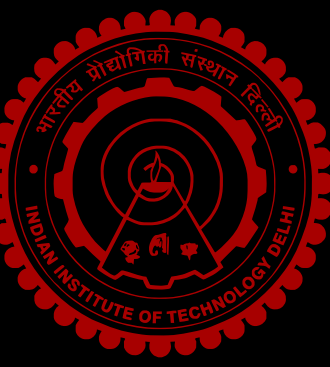
$$M_{ij} = \sum_{x \in ROI} \sum_{y \in ROI} x^i y^j I(x, y)$$

3. 2^{nd} order central moments: $\mu_{02} = \frac{M_{02}}{M_{00}} - y_c^2$, $\mu_{20} = \frac{M_{20}}{M_{00}} - x_c^2$, $\mu_{11} = \frac{M_{11}}{M_{00}} - x_c y_c$

4. Orientation: $\theta = \frac{1}{2} \tan^{-1} \left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right)$

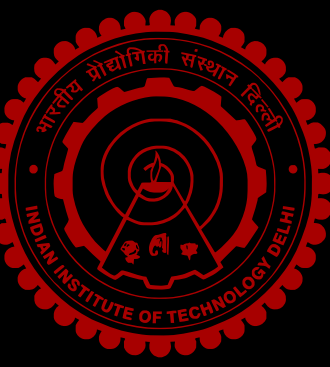
- Expand the search window (for use by the next iteration of mean-shift)

CAM-Shift Tracker



Optical Flow

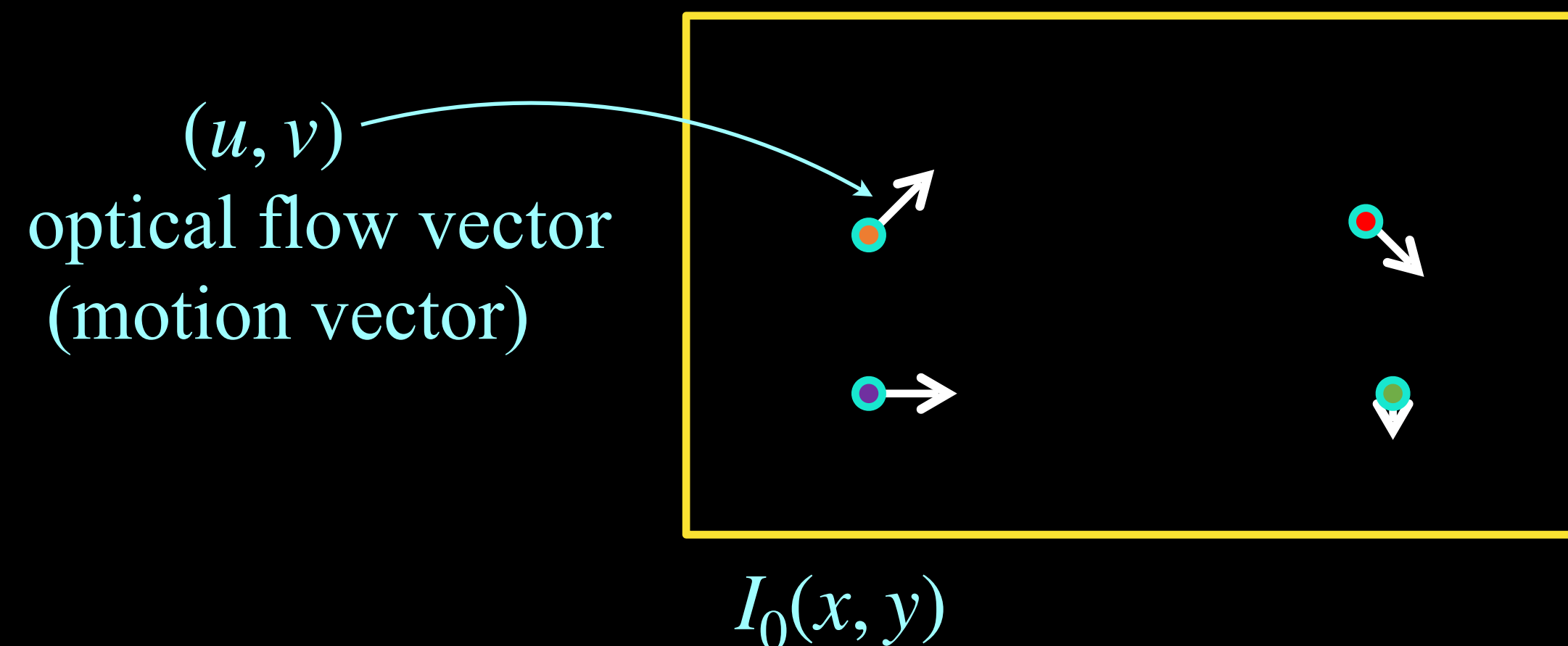
Pixel-wise



Differential Tracking using Optical Flow

Tracks the motion of pixels in consecutive frames

- Motion vector at each pixel
 - can be due to object and camera movement

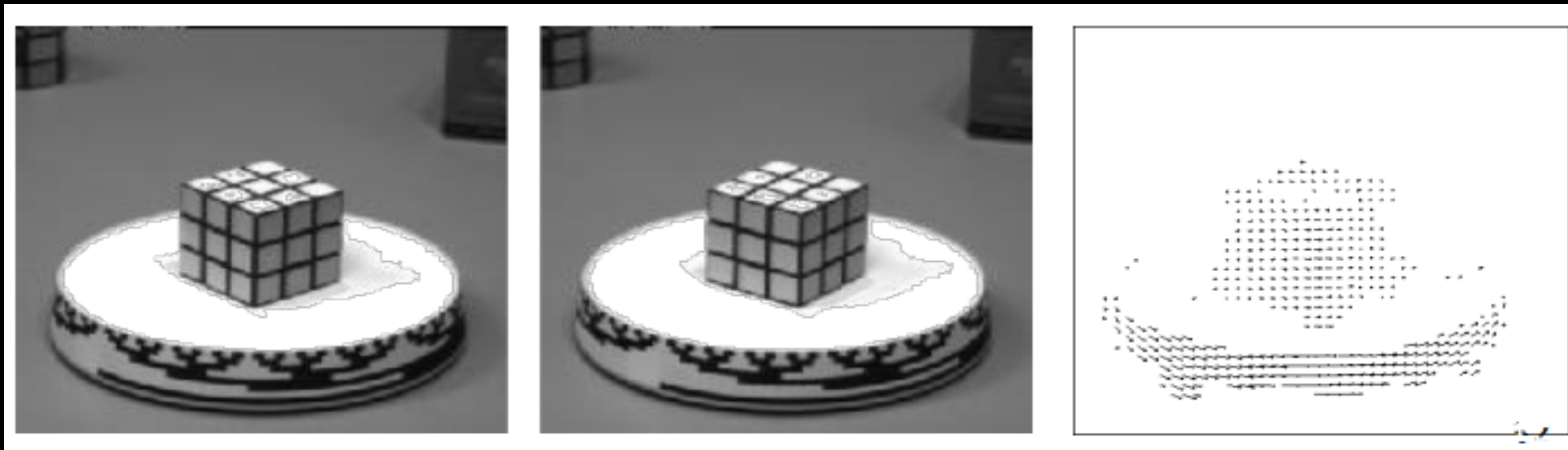


$\Rightarrow$ Point at pixel $I_0(x, y)$ moves to $(x+u, y+v)$ in I_1
i.e., $I_0(x, y)$ is 'the same as' $I_1(x+u, y+v)$

Differential Tracking using Optical Flow

Tracks the motion of pixels in consecutive frames

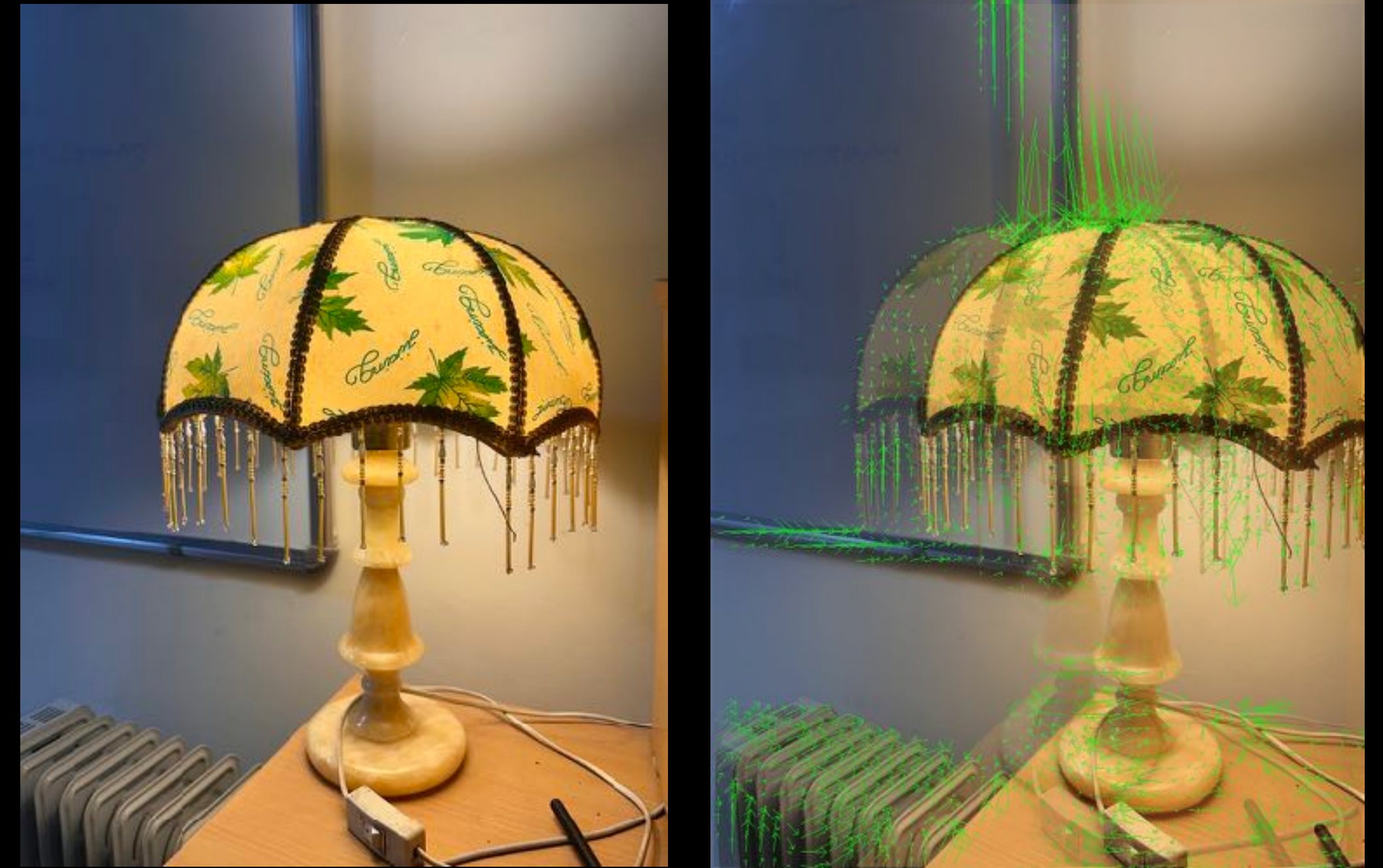
- Motion vector vector at each pixel
 - can be due to object and camera movement

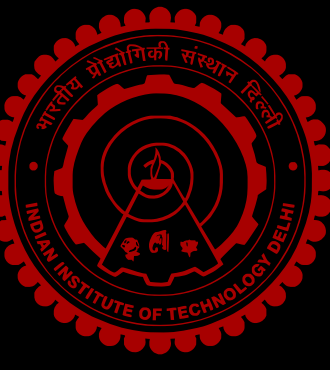


$\Rightarrow$ Point at pixel $I_0(x, y)$ moves to $(x+u, y+v)$ in I_1
i.e., $I_0(x, y)$ is 'the same as' $I_1(x+u, y+v)$

Assumptions in Optical Flow Computation

- Temporal coherence
 - Small motion between frames (~ 1 -2 pixels)
- Photo constancy
 - Color intensities remain similar
- Spatial coherence
 - neighboring pixels in a local window have similar motion, likely belonging to the same object (neighborhood consistency)





Point Tracking with Differentiable Methods

- A pixel of interest (x, y) in the first image I_0 is similar to the one at $(x+u, y+v)$ in the subsequent image I

- Find the nearest color match in a neighborhood, and assume it is the same point?

$$I(x+u, y+v) - I_0(x, y) \approx I(x, y) + I_x(x, y)u + I_y(x, y)v - I_0(x, y) \quad : \text{assuming small } (u, v), < 1 \text{ pixel?}$$

Usually Luminance

$$= \Delta I(x, y) + I_x(x, y)u + I_y(x, y)v = 0, \text{ assuming color constancy}$$

One equation, Two unknowns

- To find (u, v) , assume the same flow in a small window:

- Solve $Ax = b$ in least square sense

$$I_x(x_i, y_i)u + I_y(x_i, y_i)v = -\Delta I(x_i, y_i) \quad \forall i$$

$$\Rightarrow A^T Ax = A^T b \Rightarrow \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}$$

Aperture Problem



$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}$$

Structure tensor

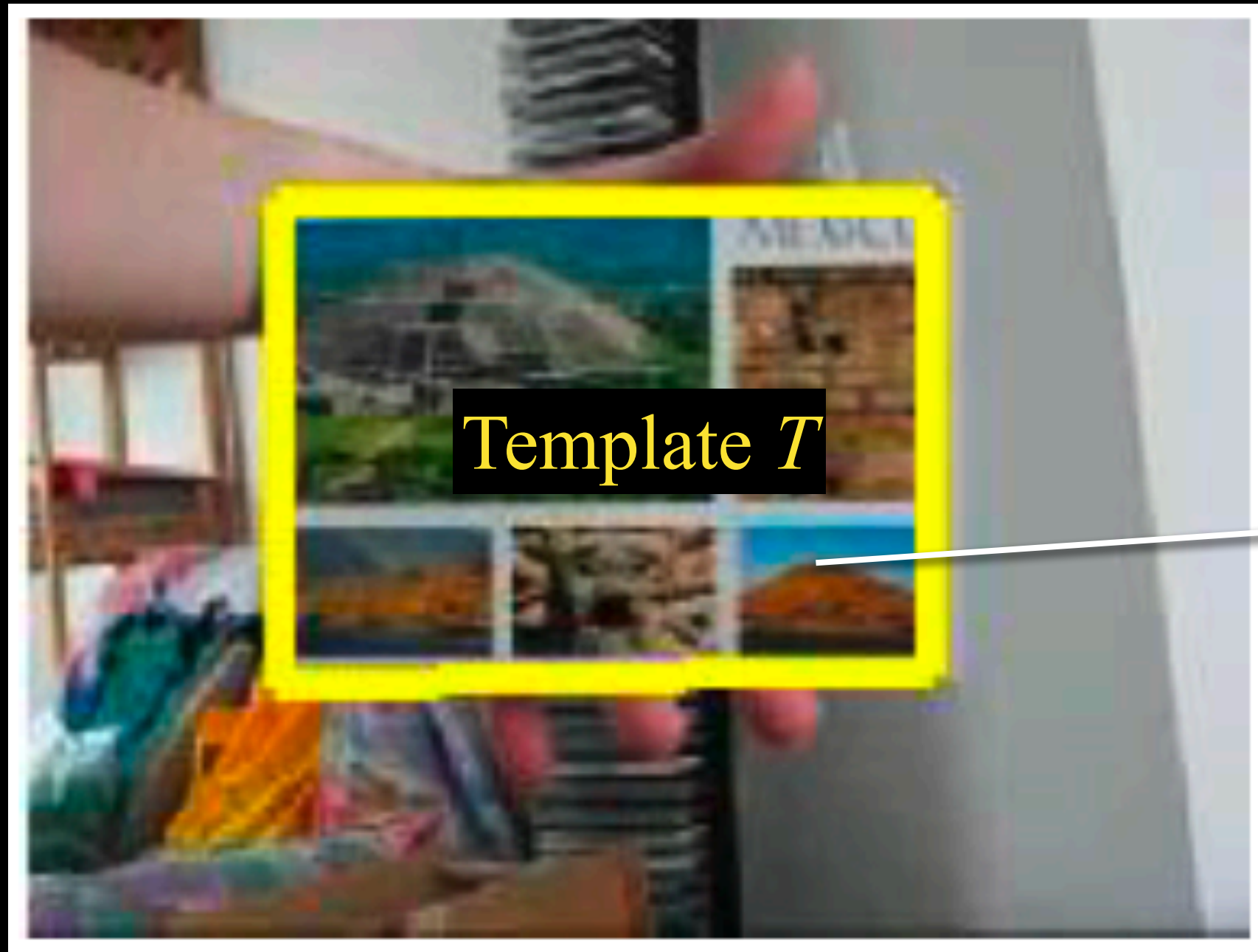
Use corner detector to select neighbors that track well

A Generalized Tracker

Template warping

Sounds like bundle-adjustment:
Predict M and improve iteratively

- Consider template sub-image T of image I_0 , and denote its points as $P_i = [x, y]^T$
- T is transformed by some warp W into the next frame I , *i.e.*, $P_i \in T \rightarrow W(P_i)$ in I

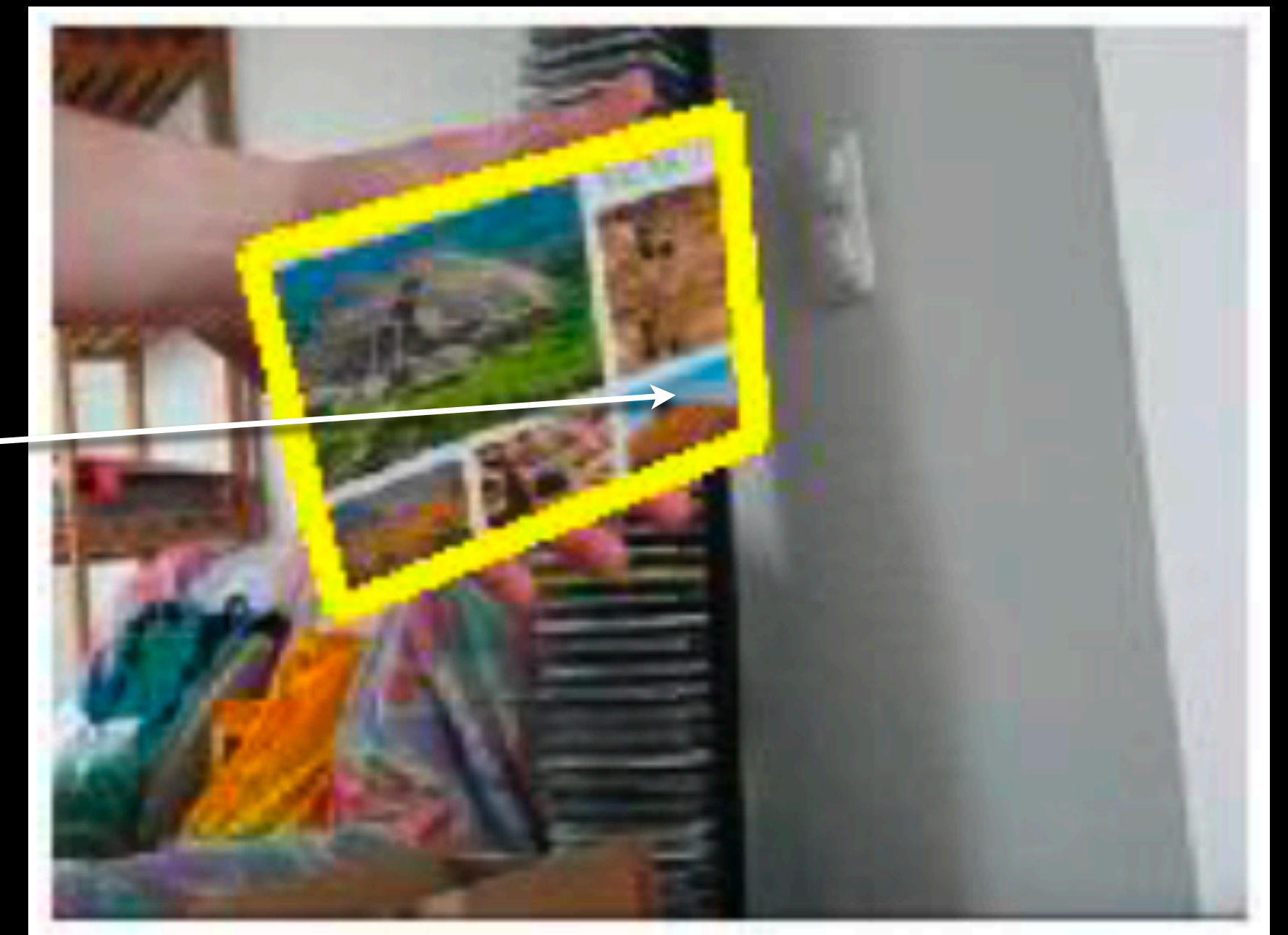


Previous image I_0

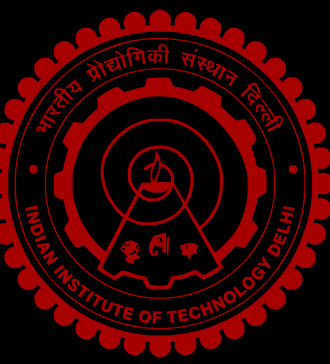
Alternate notation:
 $W(P; M)$

Warp $W(P)$

Maps pixel P in the
coordinate frame of T
to sub-pixel $W(P)$ in I



Current image I



Kanade-Lucas-Tomasi Algorithm

KLT

Recall BA: $[H + \lambda \text{diag}(J^\top J)] \Delta M = -J^\top r$

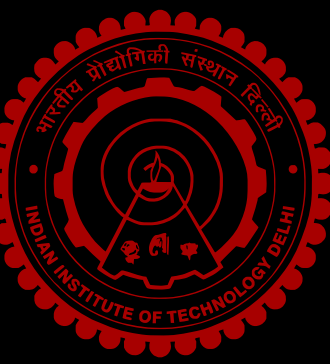
- Select good feature points P_i within Template T and start with an estimate of M (numbers representing W). Then, iterate $M + \Delta M$ until convergence:

1. For each P_i , compute $I(W(P_i))$ and the errors: $r_i = I(W(P_i)) - T(P_i)$
2. Warped gradients in I : $\nabla I_i = I_x, I_y$ at each $W(P_i)$
3. Jacobians of the warp at each point, $\bar{J}_i = [\dots \frac{\partial W(P_i)}{\partial M_j} \dots]$
4. Jacobians of $Diff$ in intensity $J_i = \nabla I_i \bar{J}_i$: chain rule gives steepest descent of I
5. Compute the Hessian $H = J^\top J = \sum J_i^\top J_i$

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \dots \frac{\partial W_x}{\partial M_j} \dots \\ \frac{\partial W_y}{\partial M_j} \dots \end{bmatrix}$$

$1 \times m$ for grayscale I

$$\Delta M = -H^{-1} J^\top r = \underbrace{-H^{-1} \sum_{m \times 1} J_i^\top}_{m \times 1} \underbrace{r_i}_{1 \times 1}$$

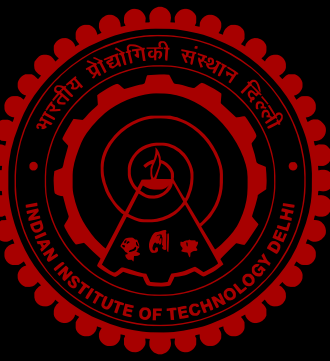


Kanade-Lucas-Tomasi Algorithm

KLT

- Select good feature points P_i within Template T and start with an estimate of M (numbers representing W). Then, iterate $M + \Delta M$ until convergence:
1. For each P_i , compute $I(W(P_i))$ and the errors: $r_i = I(W(P_i)) - T(P_i)$
 2. Warped gradients in I : $\nabla I_i = I_x, I_y$ at each $W(P_i)$

For optical flow: $M = (u, v)$,
 $W(x, y) = (x + u, y + v)$
 3. Jacobians of the warp at each point, $\bar{J}_i = [\dots \frac{\partial W(P_i)}{\partial M_j} \dots] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
 4. Jacobians of *Diff* in intensity $J_i = \nabla I_i \bar{J}_i \longrightarrow J = \begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \end{bmatrix}$
 5. Compute the Hessian $H = J^\top J = \sum J_i^\top J_i$
- $$\Delta M = -H^{-1} J^\top r = -H^{-1} \sum J_i^\top r_i \longrightarrow \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum I_x e \\ \sum I_y e \end{bmatrix} \quad (\text{Let } e_i = -r_i)$$



Handling Large Changes

- Assumptions invalidated if an object moves too quickly
- KLT tracker solves this by building a Gaussian image pyramid
 - Gaussian filtered 1/2-sized, 1/4-sized, ...
- Starts the optical flow computation at the coarsest level, where u, v are small (in that pixel space)
- Passes the computed u, v to the next finer level to use as a starting guess
- Still can have problems with large changes, quality of features, initialization, occlusion, illumination changes, drift ...