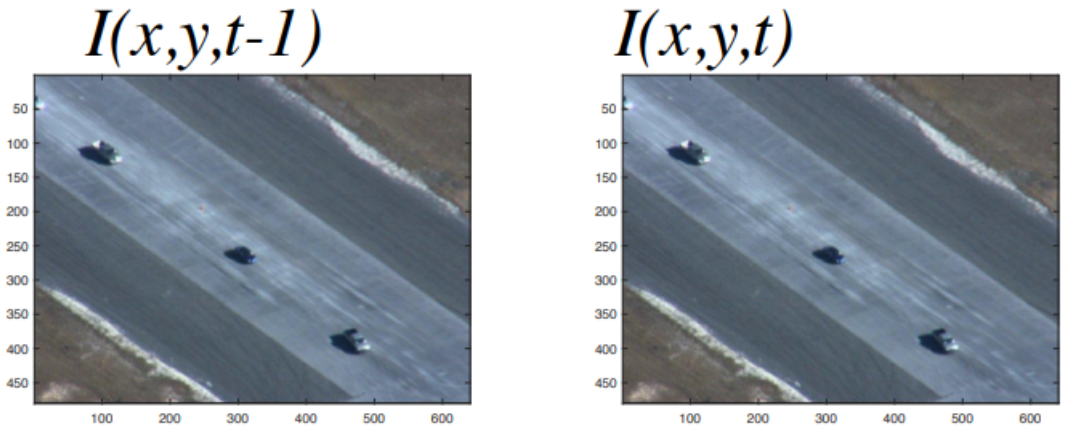
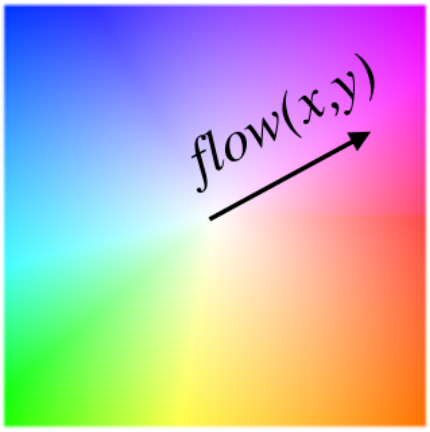
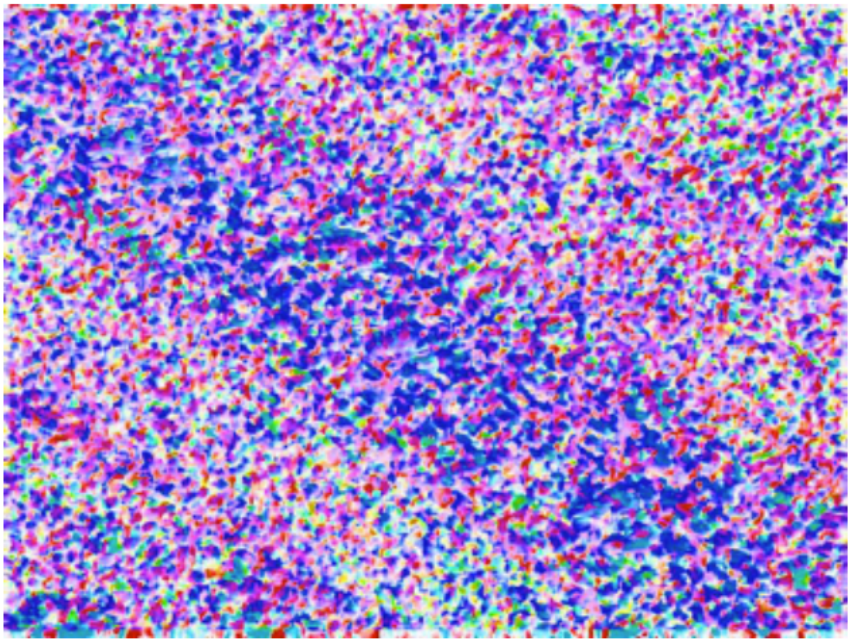
Tracking

Pixel Tracking



Recall that in Optical Flow we are able to calculate the velocity of all pixels in a picture. Using this velocity we could track a single point / all pixels.





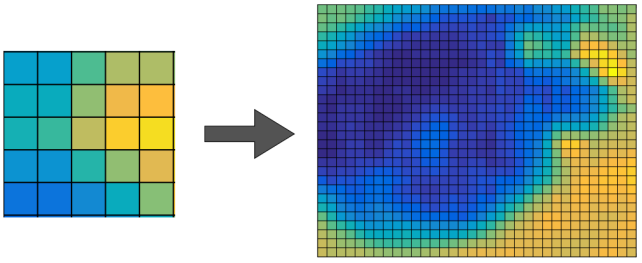
Template Tracking

Generalized Lucas-Kanade Tracking & Kalman Filter

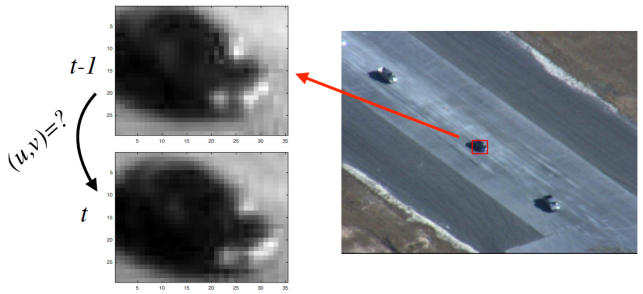
Lucas-Kanade Template Tracker

We now want to track a special box instead of a single pixel

- Replace 5×5 window with user-specified template window



- Compute flow-vector per template, not per pixel



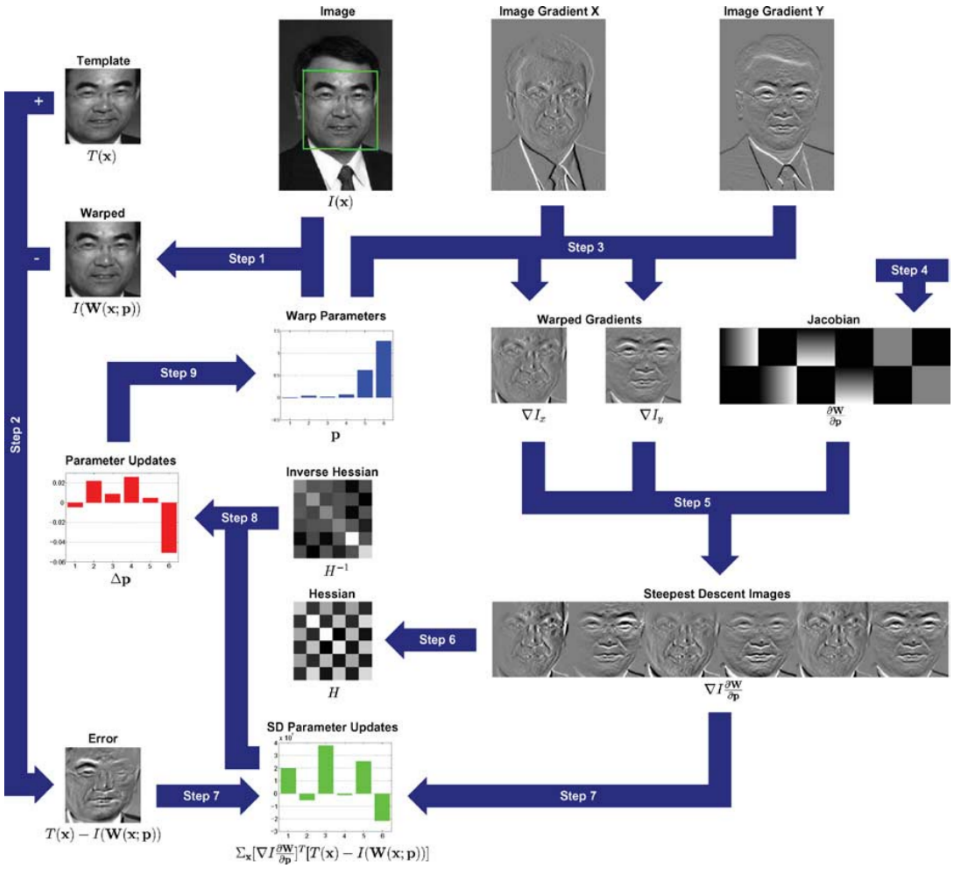
- Sum over pixels inside template-window

$$
\begin{aligned}
E(u,v) &= \sum_{\mathbf{x}} [I(x+u, y+v) - T(x,y)]^2 \\
&\approx \sum_{\mathbf{x}} [I(x,y) + uI_x(x,y) + vI_y(x,y) - T(x,y)]^2 \\
&= \sum_{\mathbf{x}} [uI_x(x,y) + vI_y(x,y) + D(x,y)]^2 \quad \text{with } D = I - T
\end{aligned}
$$

and solve

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

Generalized LK Template Tracker



The problem with the simple Lucas-Kanade Template Tracker is that it assumes pure translation for all pixels in a larger window, which is unreasonable for long periods of time.

We now allow arbitrary template transformation-model instead of only pure translation

$$E(u,v) = \sum_{\mathbf{x}} [I(x+u, y+v) - T(x,y)]^2$$
$$\Downarrow$$
$$E(\mathbf{p}) = \sum_{\mathbf{x}} [I(\mathbf{W}([x,y];\mathbf{p})) - T([x,y])]^2$$

where $\mathbf{p}$ is the warp parameter (translation, rotation, affine, etc.)

To optimize the warped version, the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\mathbf{p}$; i.e. the following expression is (approximately) minimized:

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

w.r.t $\Delta\mathbf{p}$ and then the parameters are updated:

$$\mathbf{p} \Leftarrow \mathbf{p} + \Delta\mathbf{p}$$

The Lucas-Kanade algorithm (which is a Gauss-Newton gradient descent non-linear optimization algorithm), see Newton's method.

$$\begin{aligned}
&\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x})]^2 \\
\underset{\text{Tayler expansion}}{\Rightarrow} &\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 \\
\underset{\text{Take derivative}}{\Rightarrow} &\sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] \\
\underset{\text{Set derivative to zero}}{\Rightarrow} &\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \\
&\text{Where } H = \sum_{x} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]
\end{aligned}$$

> **The Lucas-Kanade Algorithm**
> Iterate:
> 1. Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
> 2. Compute the error image $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
> 3. Warp the gradient ∇I with $\mathbf{W}(\mathbf{x};\mathbf{p})$
> 4. Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
> 5. Compute the steepest descent image s $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
> 6. Compute the Hessian matrix
> 7. Compute $\sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
> 8. Compute $\Delta\mathbf{p}$
> 9. Update the parameters $\mathbf{p} \Leftarrow \mathbf{p} + \Delta\mathbf{p}$
> until $\|\Delta p\| \leq \epsilon$

Pros:

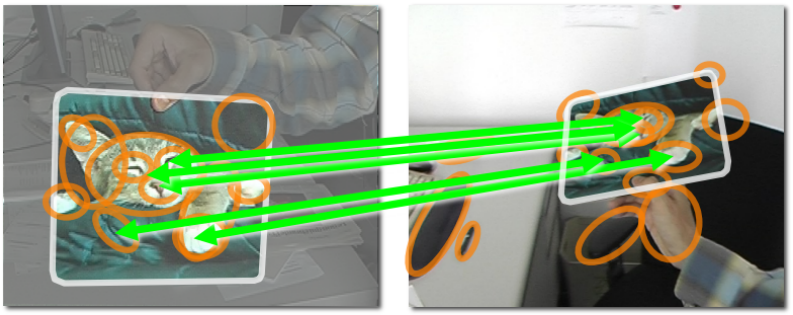
- can handle different parameter space
- can converge fast in a high-frame rate video

Cons:

- not robust to image noise or large displacement
- some transformations are impossible to parameterize

Track by Matching

We already know how to extract features, i.e. interesting points or regions. We could use a template and find its interesting points and match them to every frame of the video to find it.



Reference image(s) of the object to detect

Test image

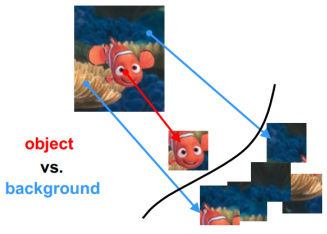
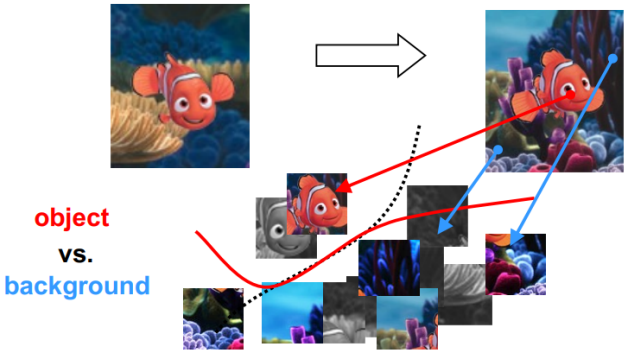
Track by Detection

If an object is in the detection database, we could detect it in each frame independently to achieve tracking.

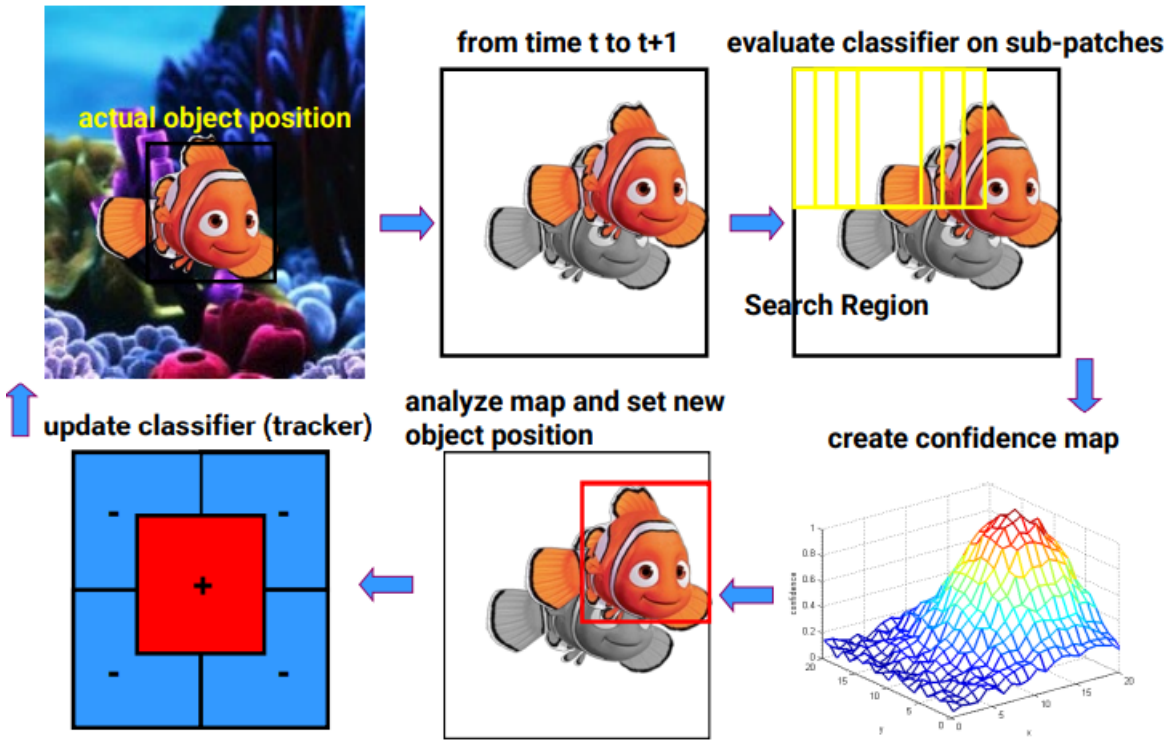
- Detect object(s) independently in each frame
- Associate detections over time into tracks

Online Learning

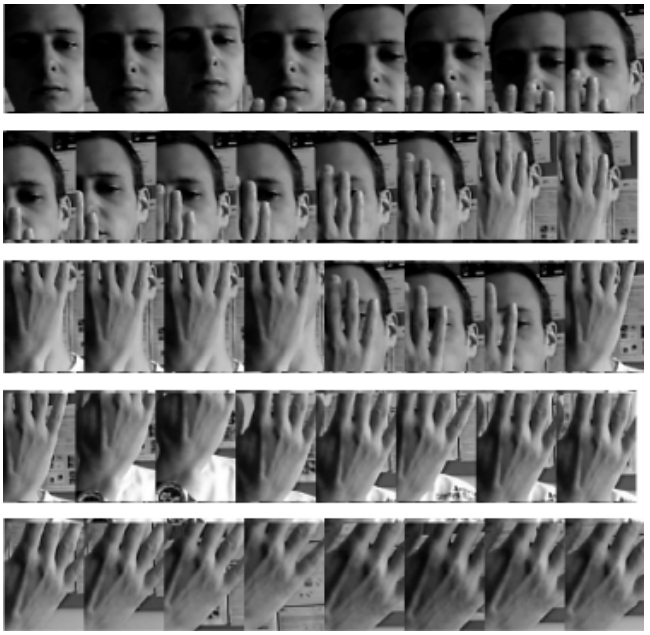
The idea is, we train an object-background classifier in the fly, i.e. update the classifier every frame.

Frame t	Frame $t+1$
	

The tracking loop is as follows



The problem this online learning has is that if we mask the object to be tracked gradually with another object, the tracker will end up tracking the masking object. This is called *gradual drift*.



In the above example, the tracker will finally track the hand instead of the face. To avoid drift, we can "anchor" our model with the initial object in order to help avoid drift.