Computer Vision
(Summer Semester 2022)

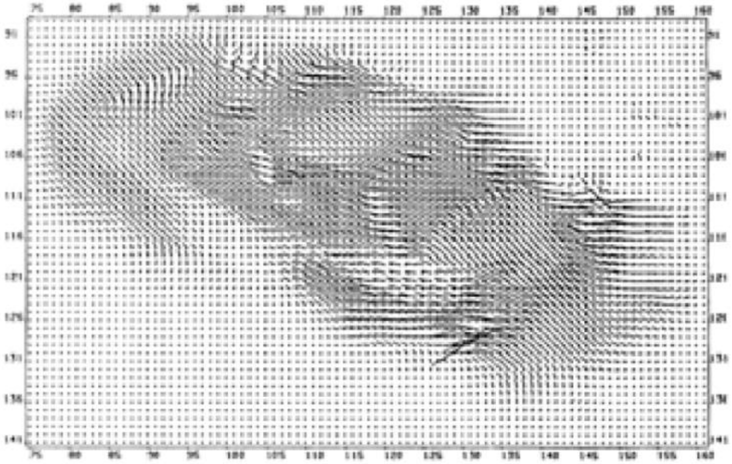
Lecture 8

Dense Motion Estimation

- Slides based on
  - Szeliski’s book “Computer Vision: Algorithms and Applications”: http://szeliski.org/Book/
  - Slides from Brown University: https://cs.brown.edu/courses/csci1430/
  - Slides from Elli Angelopoulou from summer term 2014: https://www5.cs.fau.de/lectures/ss-14/computer-vision-cv/

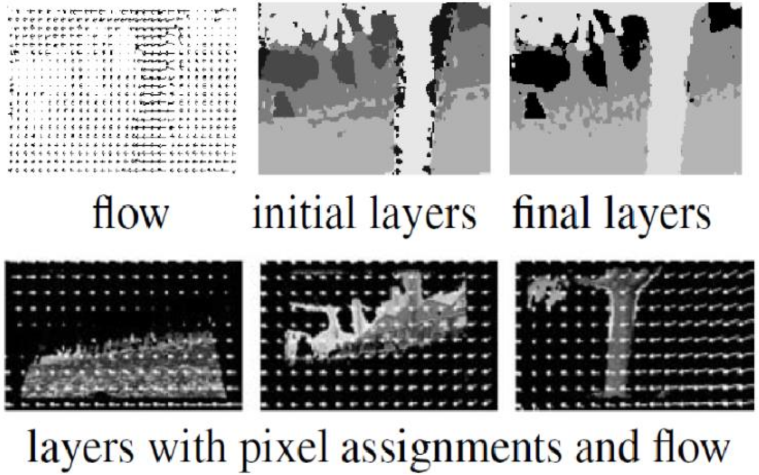
Dense Motion Estimation

- Estimate the “flow” of objects visible in single pixels from two following frames of an animation
  → flow coming from object motion



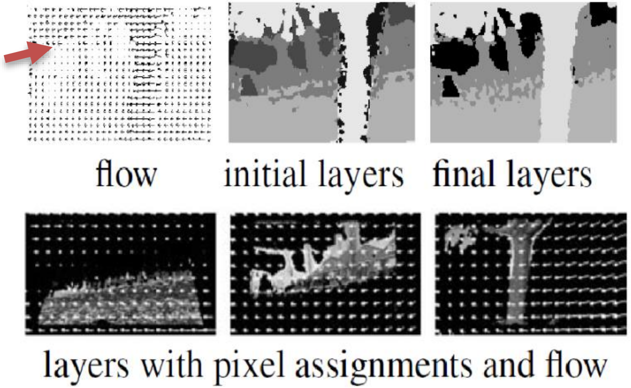
Dense Motion Estimation

- Estimate the “flow” resulting from camera movement
  → magnitude of flow related to depth
  → depth estimator



Dense Motion Estimation

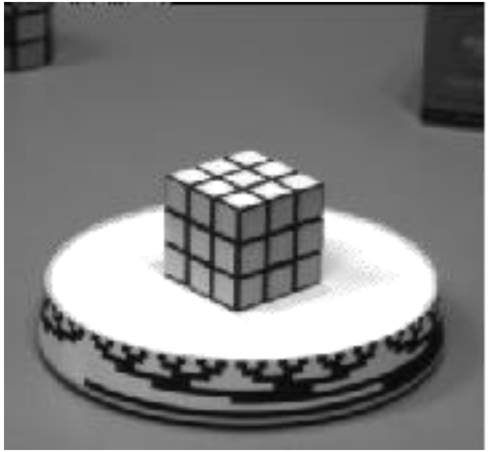
- Estimate the “flow” resulting from camera movement
  → magnitude of flow related to depth
  → depth estimator

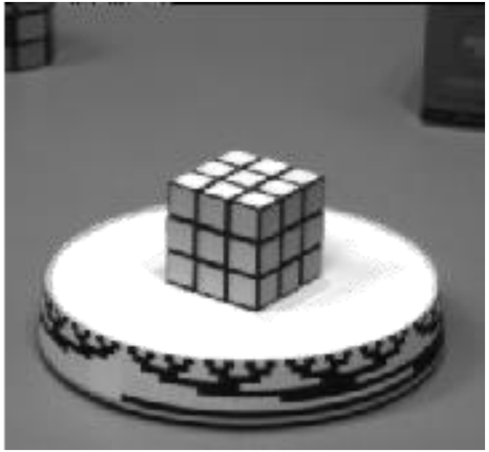


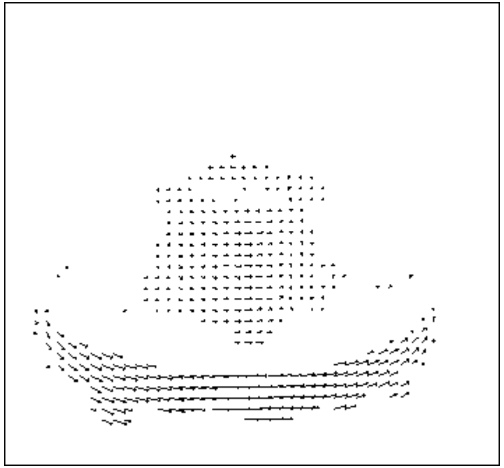
- “Dense”: estimate flow **for all** pixels, not only for certain features

Motion estimation: Optical flow

- *Optical flow* is the **apparent** motion of objects or surfaces



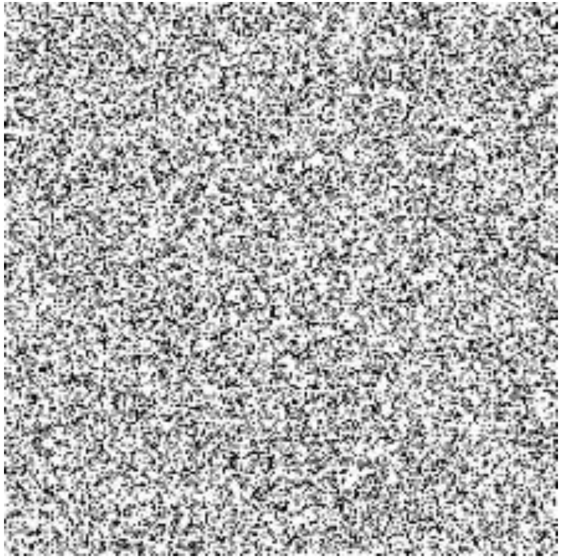




- **Optical flow only approximates motion flow**:
  white plate has no features and thus no optical flow

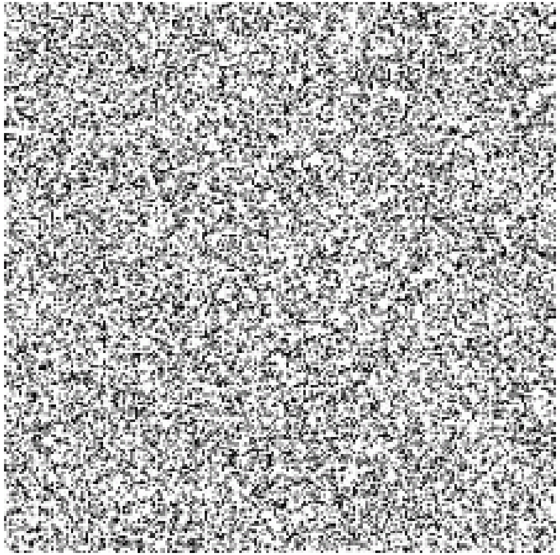
Motion and perceptual organization

- Sometimes, motion is the only cue



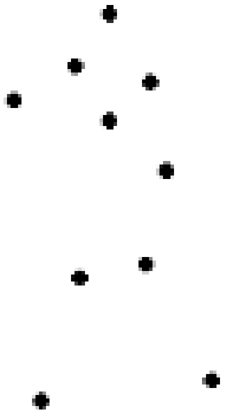
Motion and perceptual organization

- Sometimes, motion is the only cue



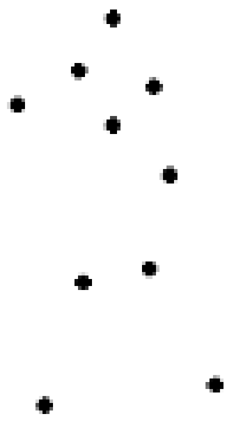
Motion and perceptual organization

- Even “impoverished” motion data can evoke a strong percept

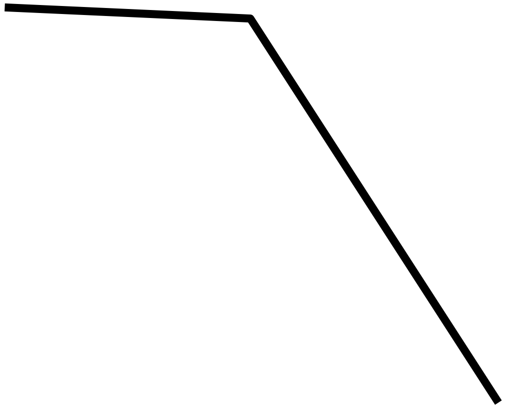


Motion and perceptual organization

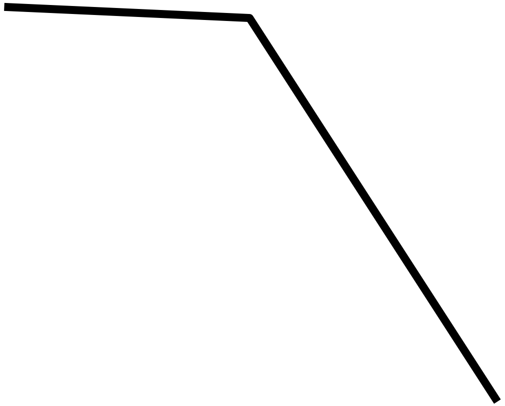
- Even “impoverished” motion data can evoke a strong percept



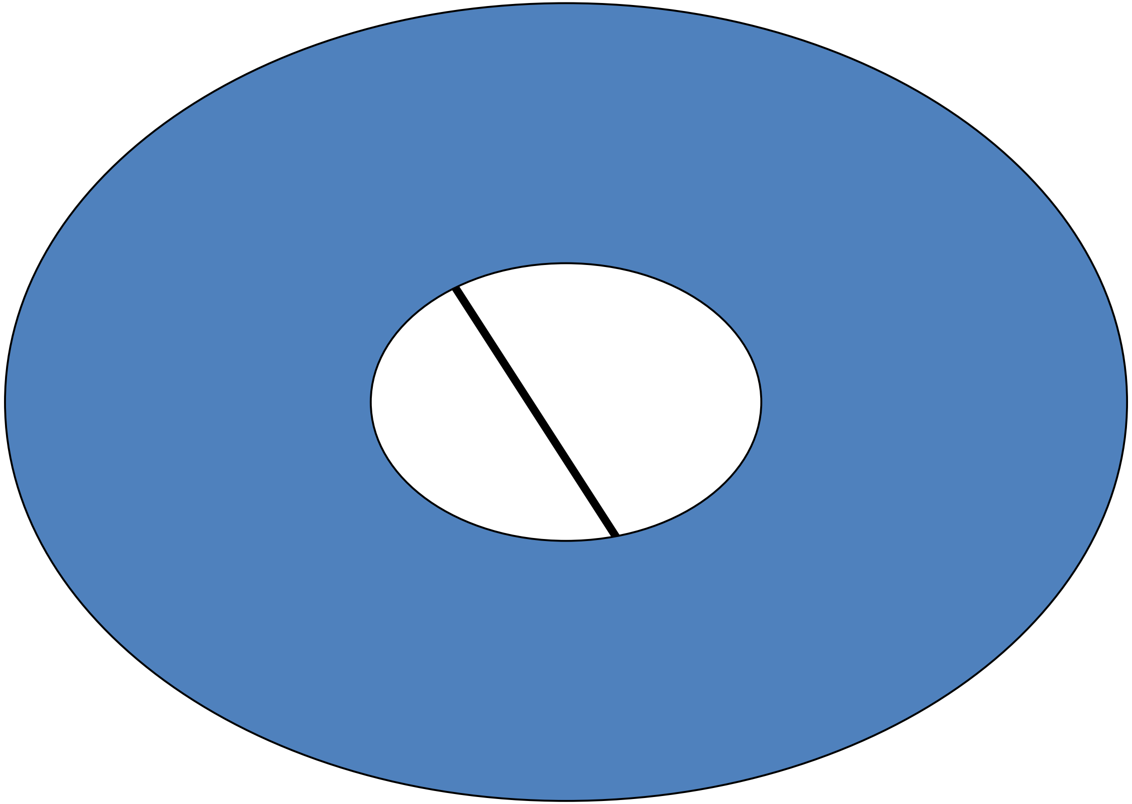
Aperture problem



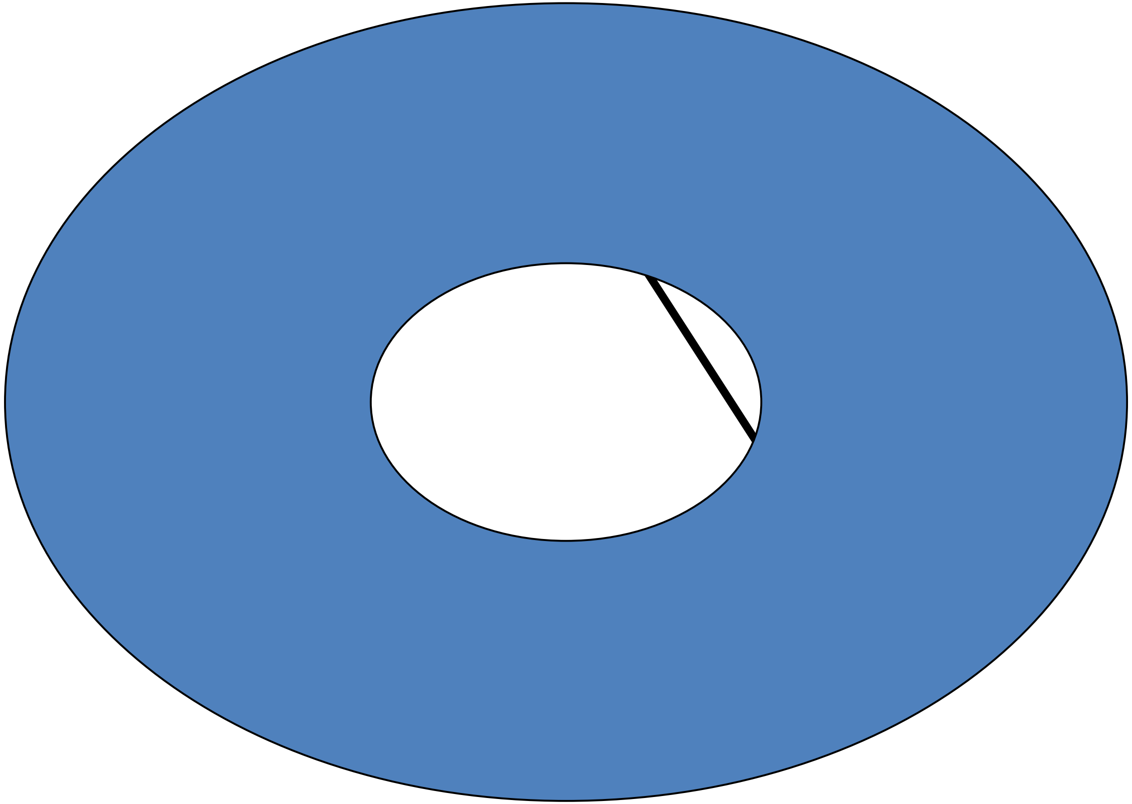
Aperture problem



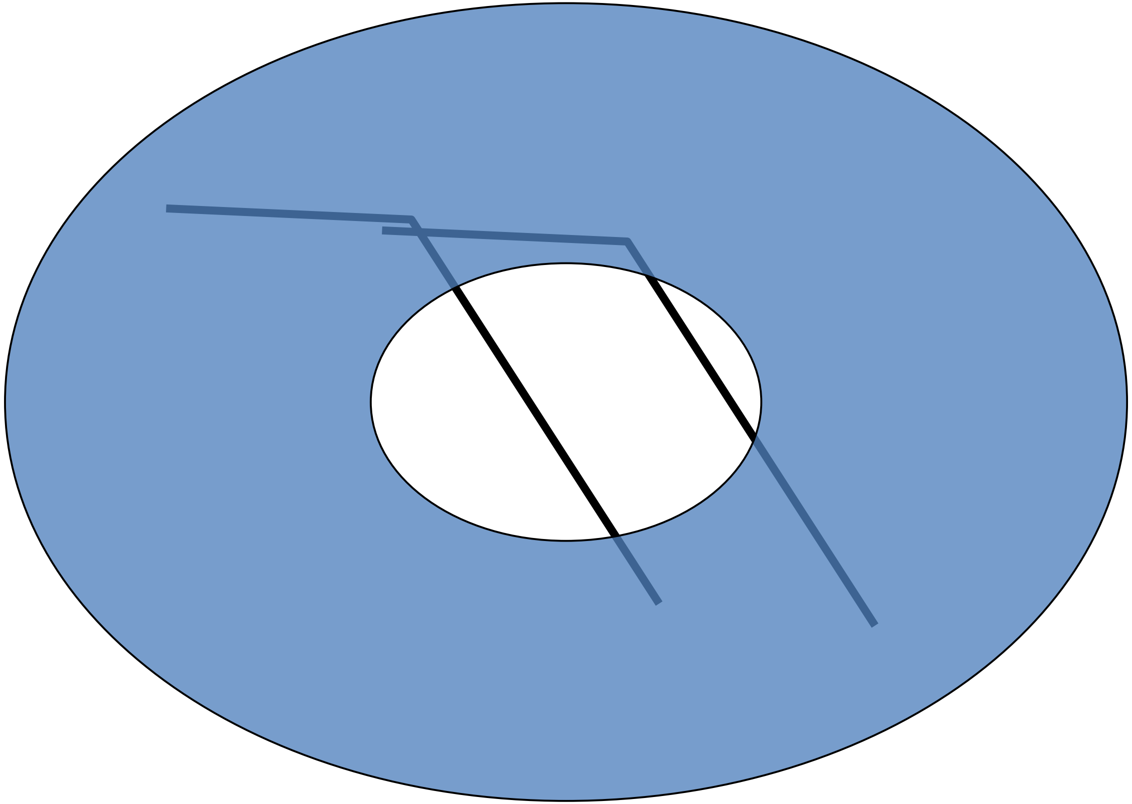
Aperture problem



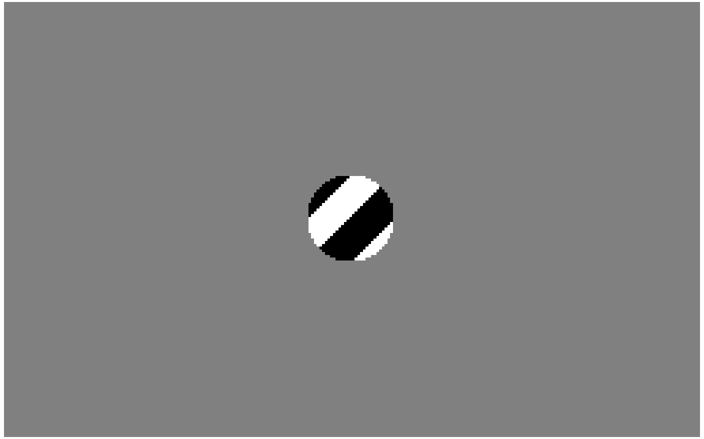
Aperture problem



Aperture problem

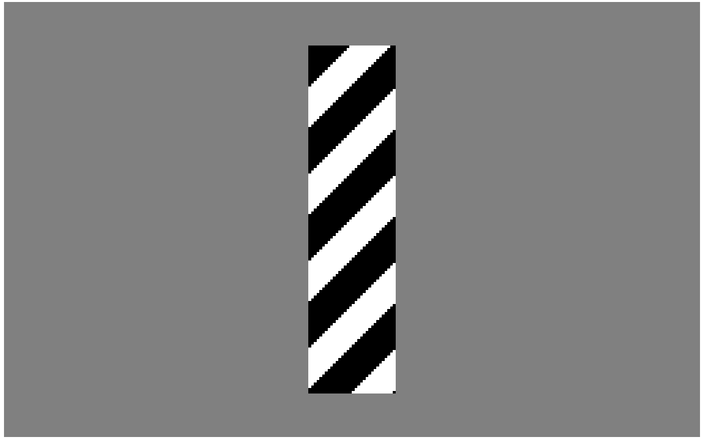


The barber pole illusion



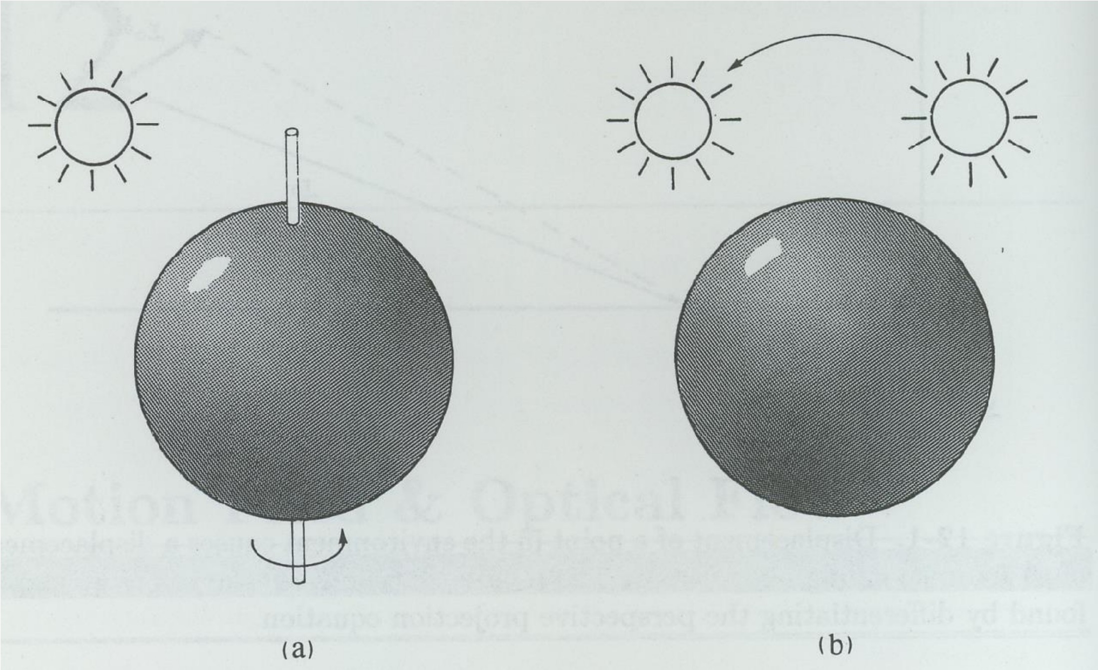
http://en.wikipedia.org/wiki/Barberpole_illusion

The barber pole illusion



http://en.wikipedia.org/wiki/Barberpole_illusion

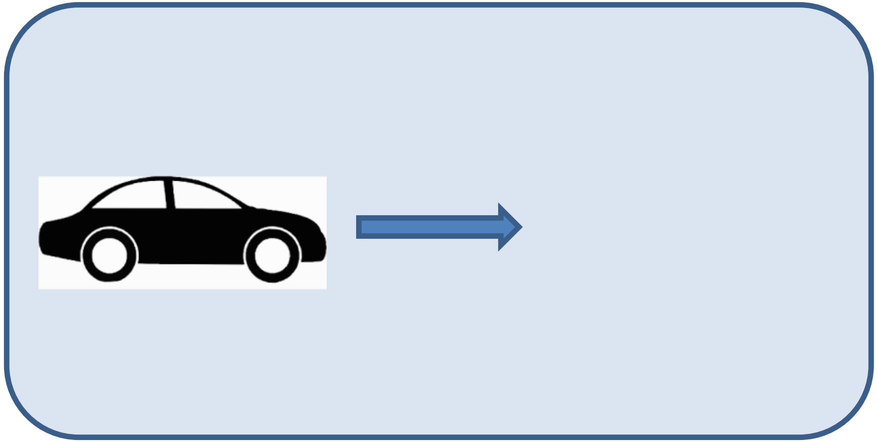
Motion flow vs. optical flow



left: sphere rotates → motion flow, but no optical flow
right: light moves → highlight moves → optical flow, but no motion flow

Motion Flow vs. Optical Flow

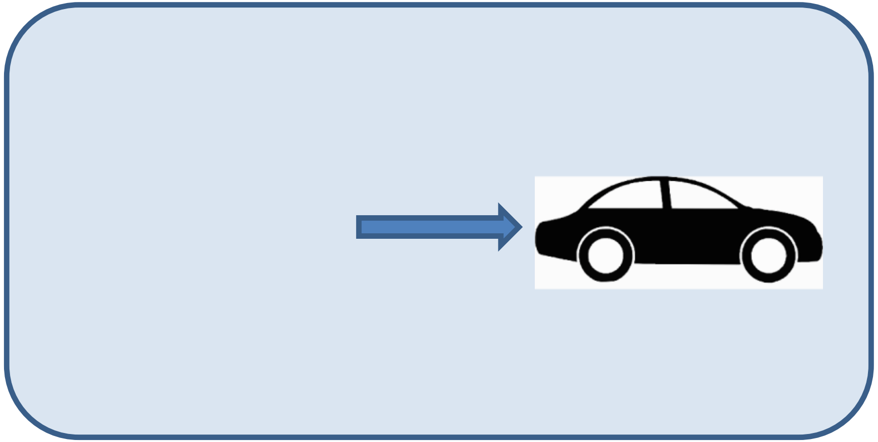
- Often used assumption: translation parallel to image plane



- 3D translation in world → 2D translation in image space

Motion Flow vs. Optical Flow

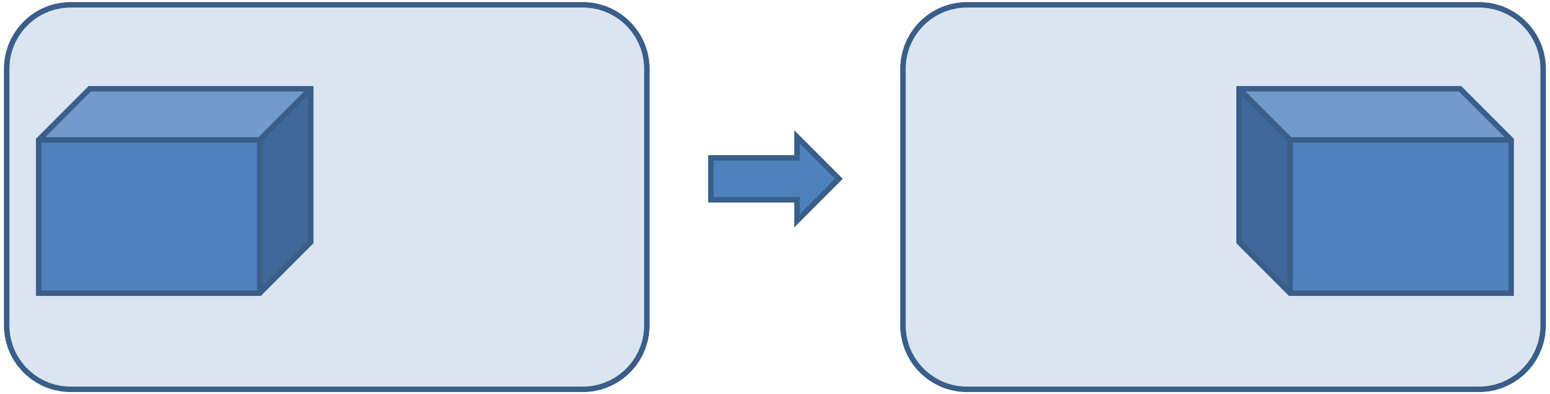
- Often used assumption: translation parallel to image plane



- 3D translation in world → 2D translation in image space

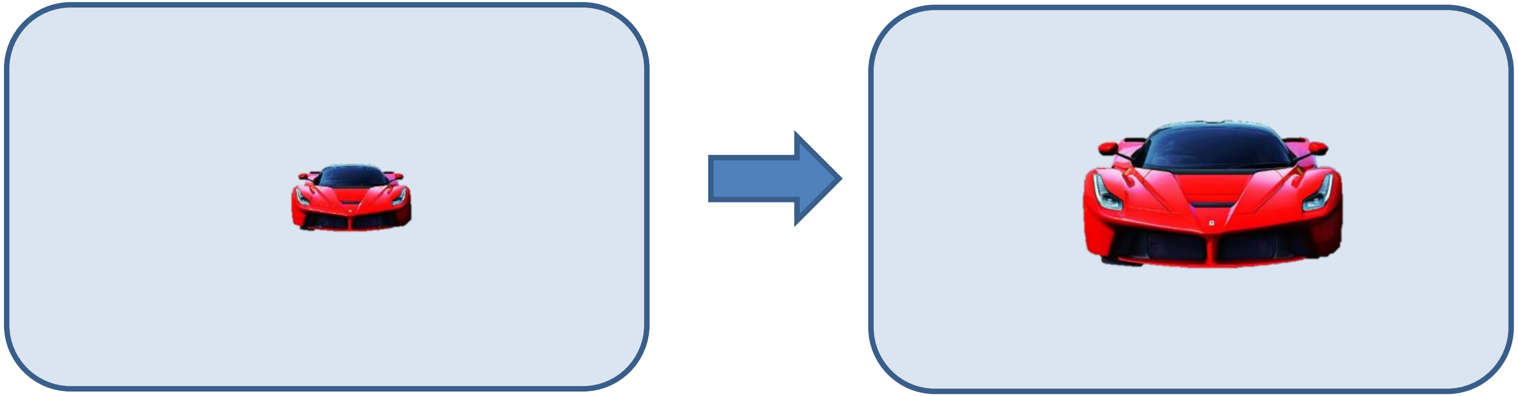
Motion Flow vs. Optical Flow

- Not really true: ignores parallax effects
- Closer objects move faster in image space than distant ones
- True for translational movement of object or camera



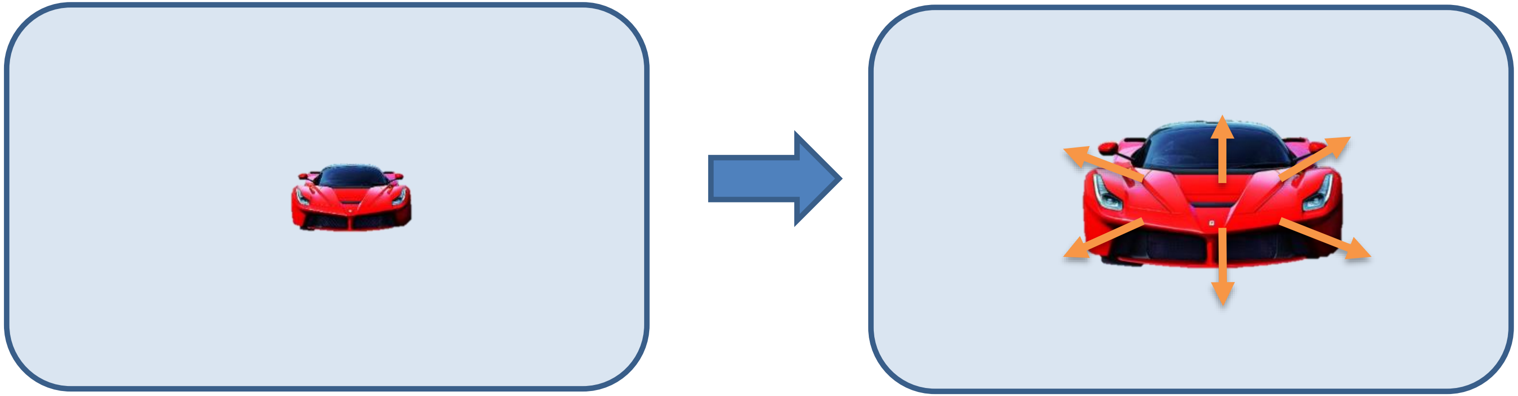
Motion Flow vs. Optical Flow

- Motion towards the camera becomes a radial optical flow



Motion Flow vs. Optical Flow

- Motion towards the camera becomes a radial optical flow



Motion Flow vs. Optical Flow

- Finally: camera rotate → additional distortion

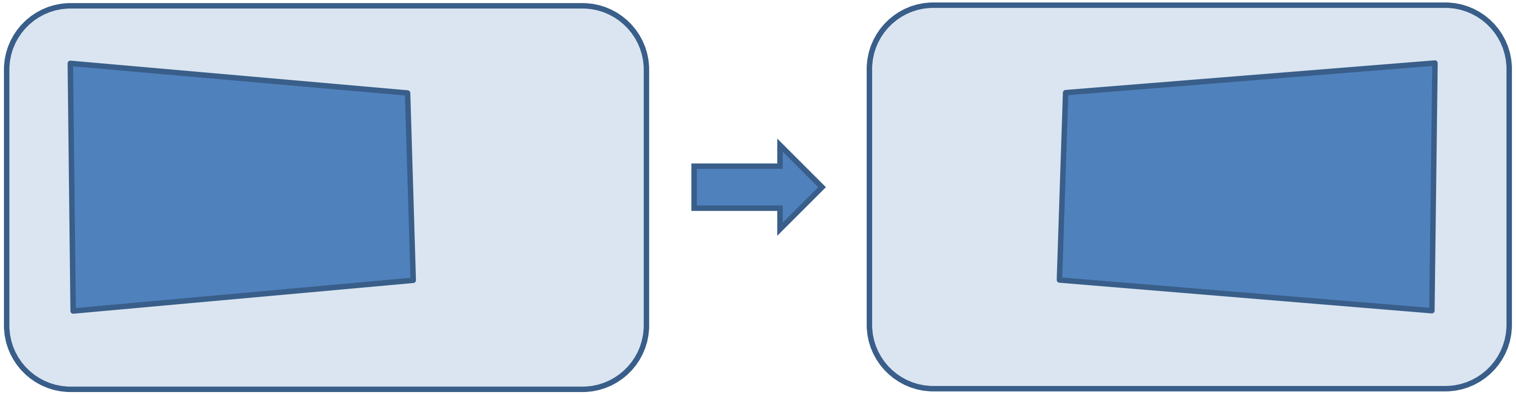
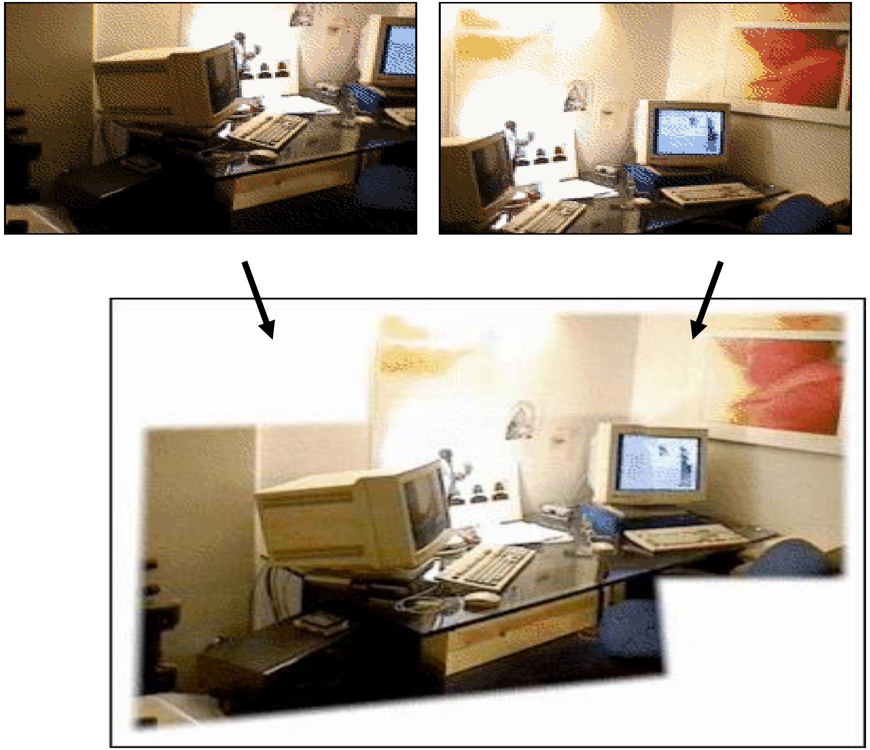
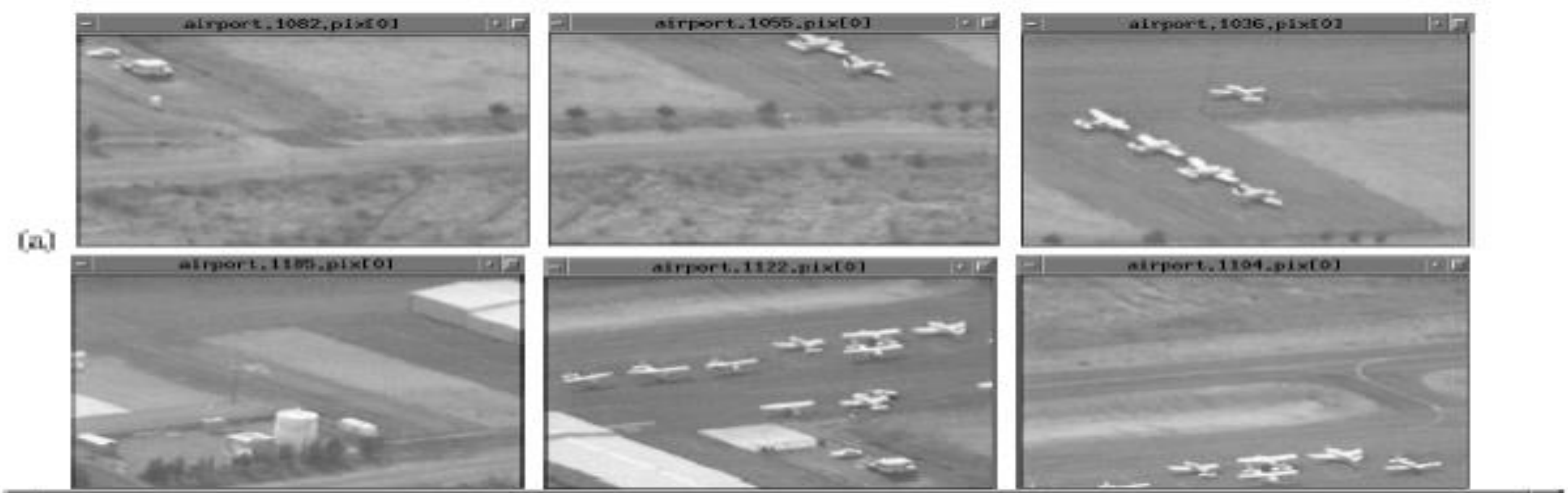


Image Alignment

- Goal: Estimate a *single* **v** translation (transformation) for the entire image.
- The entire image has the same translation value so the optical flow values for every pixel is the same.
- This is typically an easier problem than general motion estimation.
- We can compute it very well with pyramid-based methods like the Lucas-Kanade one (see later)



Mosaicing – input images



Mosaicing – Final Result

(b)

g. 1. Static background mosaic of an airport video clip.
(a) A few representative frames from the minute-long video clip. The video shows an airport being imaged from the air with a moving camera. The scene itself is static (i.e., no moving objects). (b) The static background mosaic image which provides an extended view of the entire scene imaged by the camera in the one-minute video clip.

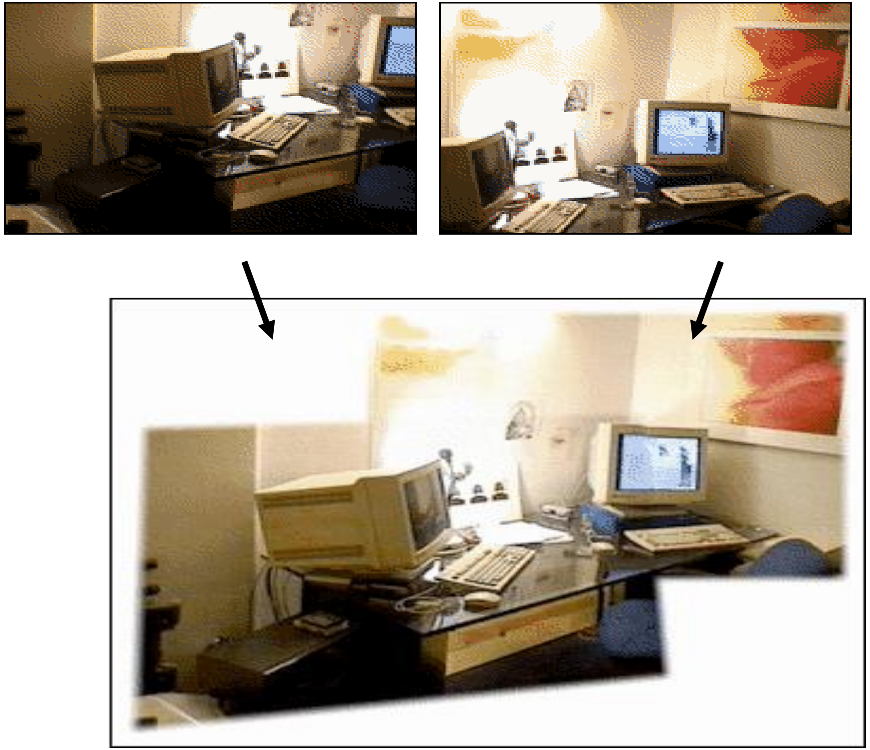
Translational Motion

- Entire image moves, e.g. from camera shake, walking camera man etc.
- Simplified assumption: all pixels move by same amount

- More realistic:
  - translational movement of object in front of static background
  - translational movement of some objects in front of static background with translational movement
    → assumption of many video coders
  - requires separation of background and foreground object(s)

Image Alignment

- Goal: Estimate a *single* **v** translation (transformation) for the entire image.
- The entire image has the same translation value so the optical flow values for every pixel is the same.
- This is typically an easier problem than general motion estimation.
- We can compute it very well with pyramid-based methods like the Lucas-Kanade one (see later)



Error Metrics

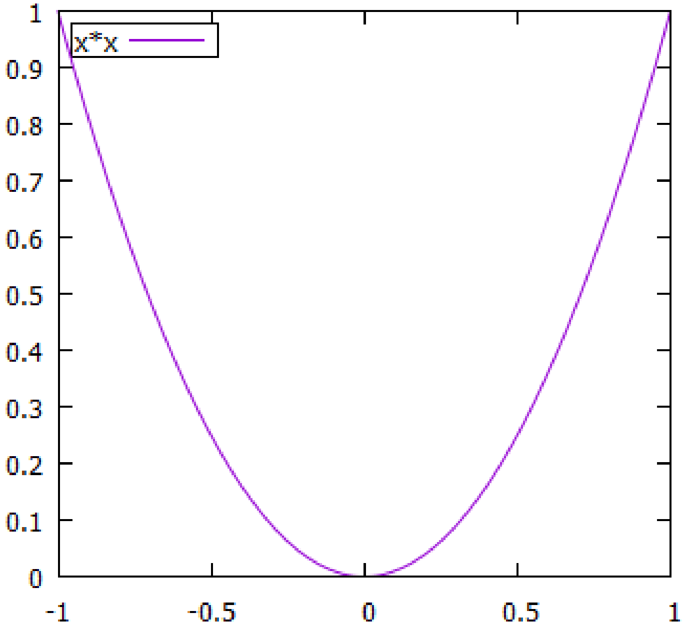
- Given two images $I_0(\boldsymbol{x})$ and $I_1(\boldsymbol{x})$ with $\boldsymbol{x}$ being the pixel position (x, y)
- We are looking for the displacement vector $\boldsymbol{u} = (u, v)$ that minimizes the difference between $I_1(\boldsymbol{x} + \boldsymbol{u})$ and $I_0(\boldsymbol{x})$
- But how can we measure the difference between two images?
- Simple solution: sum of squared difference:

$$E_{\mathrm{SSD}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \sum_i e_i^2,$$

- Color images:
  - extend sum over color channels
  - or use luminance only

Error Metrics

- SSD:
  - small errors are less important
  - larger errors strongly penalized
- → tolerant to low-amplitude noise
- later: easy for minimization !
- Downside: large errors get dominant → not very robust

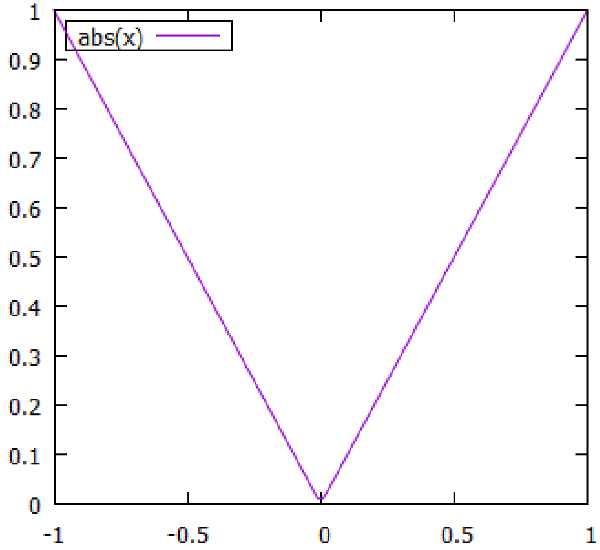


Error Metrics

- More robust: sum of absolute differences

$$E_{\mathrm{SAD}}(\boldsymbol{u}) = \sum_i |I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)| = \sum_i |e_i|.$$

- Large errors do not dominate so much



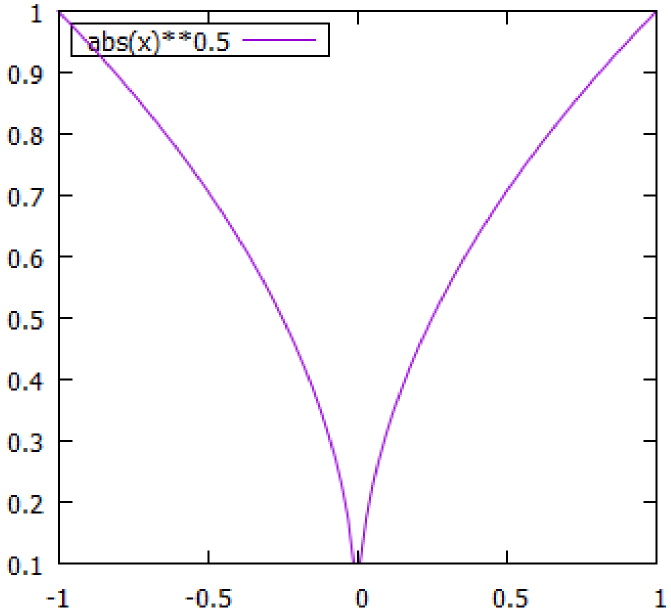
Error Metrics

- L_p-Norms:

$$E_L(\boldsymbol{u}) = \left(\sum_i |I_1(\boldsymbol{x_i} + \boldsymbol{u}_i) - I_o(\boldsymbol{x})|^p \right)^{\frac{1}{p}}$$

- In our case the exponent $\frac{1}{p}$ can be removed…
- $p = 2$: SSD
- $p = 1$: SAD

- $p < 1$ also makes sense (right plot), but is not a norm (violates triangle inequality)



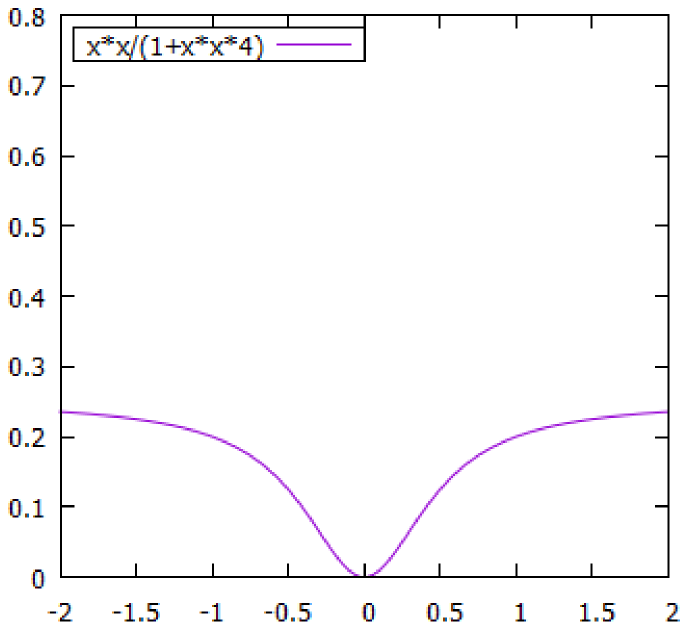
Error Metrics

- General function: robust kernel function ρ

$$E_{\mathrm{SRD}}(\boldsymbol{u}) = \sum_i \rho(I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)) = \sum_i \rho(e_i).$$

  (SRD = "sum of robust distances")
- $\rho(x) = x^p$: L_p-norm
- Problem with L_p-norm:
  not differentiable at zero
  → bad for optimization methods
- Alternative **robust** kernel functions,
  e.g. Geman-McClure kernel

$$\rho_{\mathrm{GM}}(x) = \frac{x^2}{1 + x^2/a^2}$$



FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Error Metrics

- Up to now, we ignored the fact that $I_1(\boldsymbol{x}+\boldsymbol{u})$ can be outside the image and that we also want to consider masked images
  → $w_0(\boldsymbol{x})$ and $w_1(\boldsymbol{x})$ are masks that are one inside the regions of interest
- $E_{WSRD}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x_i}+\boldsymbol{u}) w_1(\boldsymbol{x_i}) \rho\left(I_1(\boldsymbol{x_i}+\boldsymbol{u}) - I_0(\boldsymbol{x_i})\right)$
  → „W“ stands for „windowed“
- Only counts differences for pixels where both masks are one
- Possibly also normalize by area of counted pixels $A(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x_i}+\boldsymbol{u}) w_1(\boldsymbol{x_i})$:

$$E_{NWSRD}(\boldsymbol{u}) = \frac{1}{A(\boldsymbol{u})} E_{WSRD}(\boldsymbol{u})$$

Error Metrics

- Bias and Gain
  - sometimes, both images have not been taken with same exposure, aperture, or even camera
  - in this case, colors are not directly comparable, but we need a mapping, e.g. a linear one:

$$I_0(\boldsymbol{x}) \rightarrow \alpha I_0(\boldsymbol{x}) + \beta$$

  - The SSD error then becomes

$$E_{SSD}(\boldsymbol{u}) = \sum_i (I_1(\boldsymbol{x_i} + \boldsymbol{u}) - \alpha I_0(\boldsymbol{x}) - \beta)^2$$

  - We have to determine α and β such that this error is minimized → linear regression

Error Metrics

- Oftentimes a better alternative: cross correlation

$$E_{\mathrm{CC}}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u}).$$

- or even better: normalized cross-correlation

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\ [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2} \sqrt{\sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}},$$

with

$$\begin{aligned} \overline{I_0} &= \frac{1}{N} \sum_i I_0(\boldsymbol{x}_i) \quad \text{and} \\ \overline{I_1} &= \frac{1}{N} \sum_i I_1(\boldsymbol{x}_i + \boldsymbol{u}) \end{aligned}$$

- Also works when pictures were taken with different exposure !

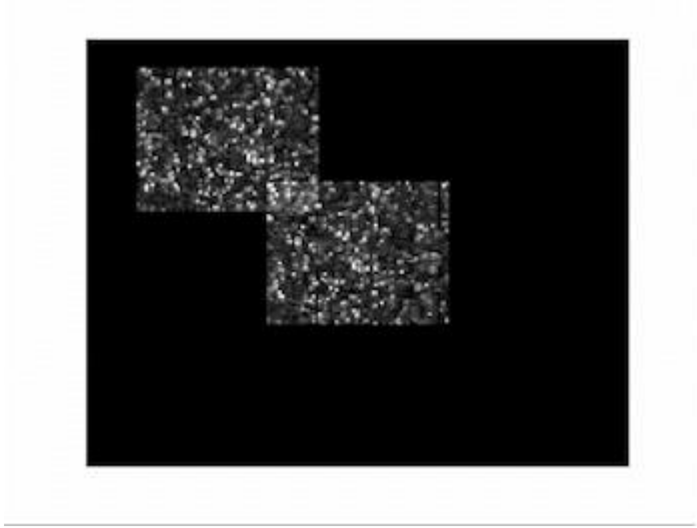
Estimation of Translational Movement

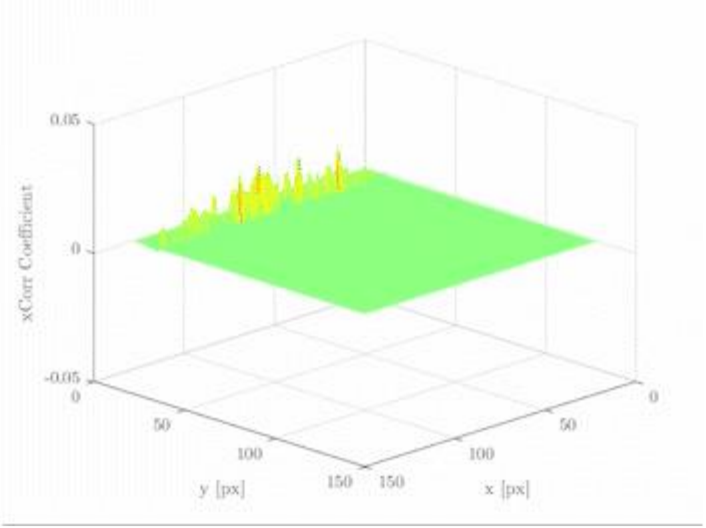
- Decide upon error metric $E(\boldsymbol{u})$
- Search for $\boldsymbol{u}$ that minimizes $E(\boldsymbol{u})$

- Full Search: examine all possible values for $\boldsymbol{u}$, maybe in a limited radius
  → very expensive
  → only on discrete grid (e.g. pixels, not subpixel-wise)

For a region of 100 x 100 in each image, there are almost 200 x 200 values to search!

Cross-Correlation + Sliding Window

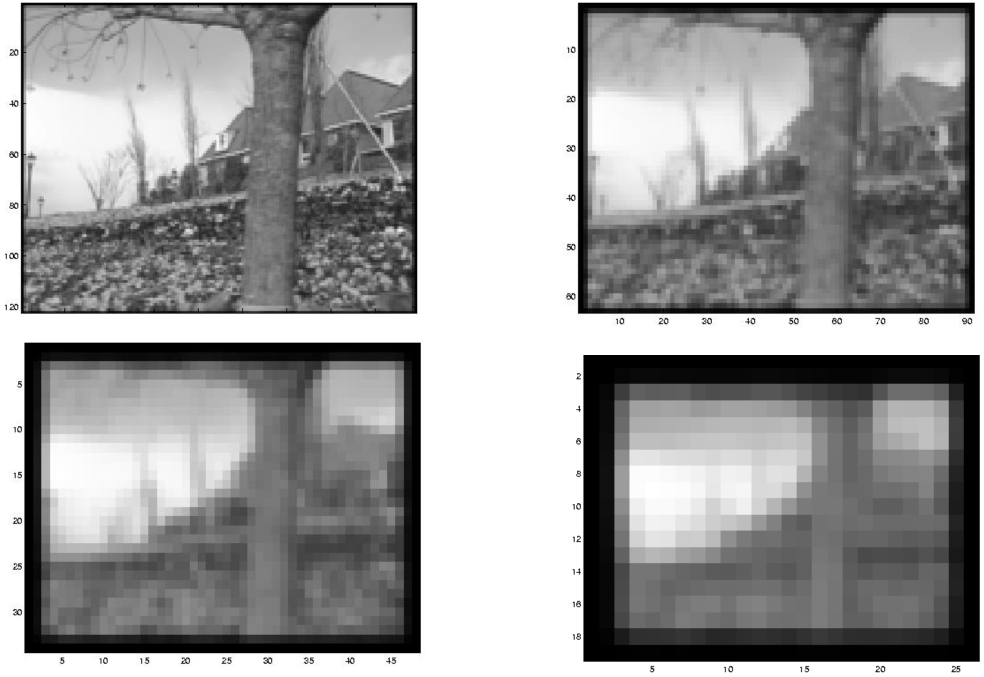




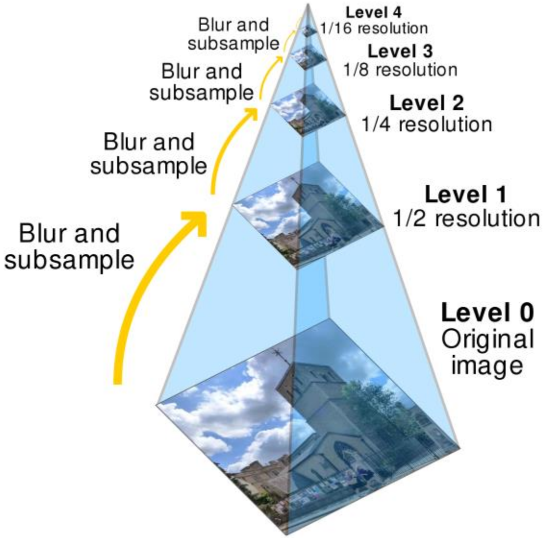
- https://en.wikipedia.org/wiki/Cross-correlation#/media/File:Cross_Correlation_Animation.gif

Estimation of Translational Movement

- **Use image pyramid**
  - starting at the coarsest resolution, we always search in a small neighbor-hood
  - This results in an initially coarse flow resolution on a small neighborhood, and expands to a fine flow resolution on a larger neighborhood



Coarse-to-fine optical flow estimation



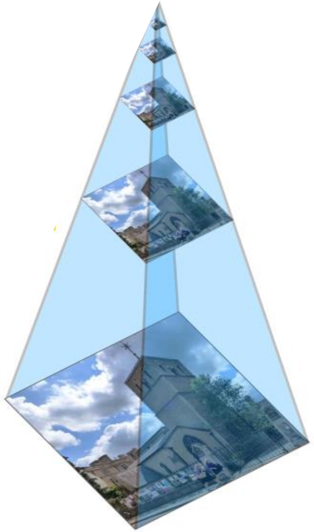


Image 1

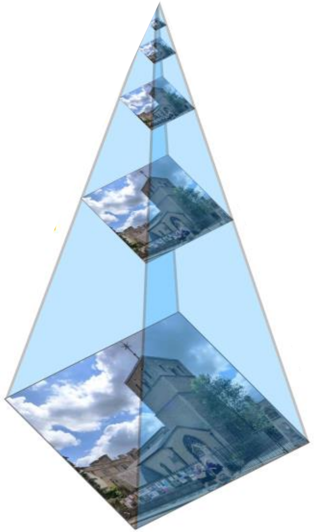
Image 2

https://en.wikipedia.org/wiki/Pyramid_(image_processing)#/media/File:Image_pyramid.svg

Gaussian pyramid of Image1

Gaussian pyramid of Image2

Coarse-to-fine optical flow estimation



u=1.25 pixels

u=2.5 pixels

u=5 pixels

u=10 pixels

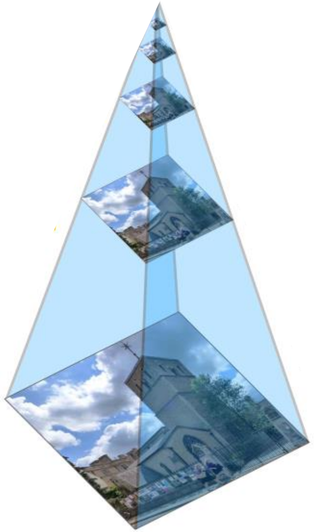


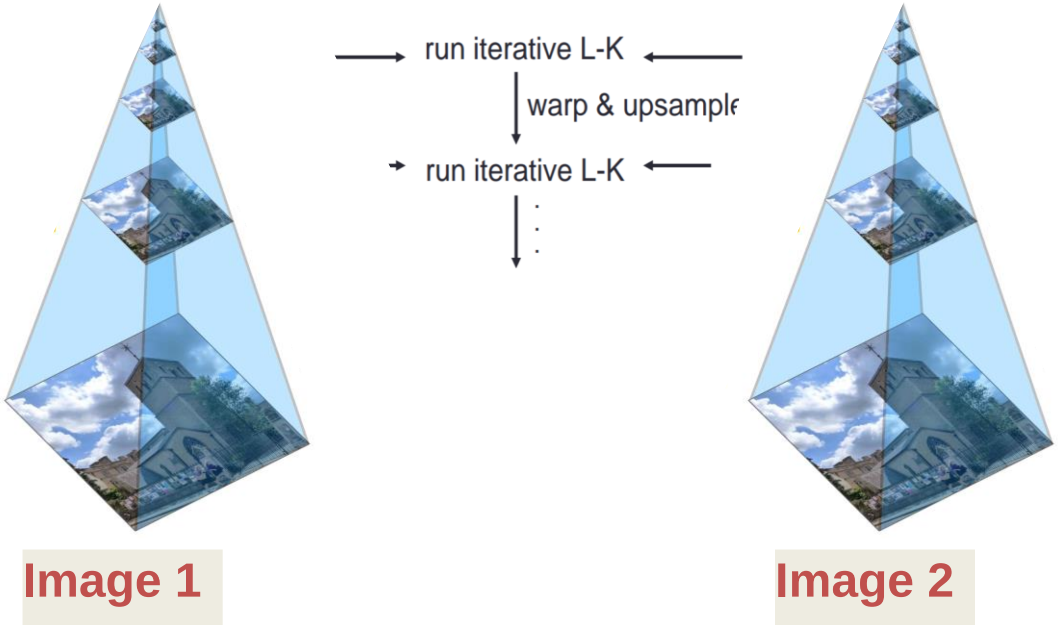
Image 1

Image 2

Gaussian pyramid of Image1

Gaussian pyramid of Image2

Coarse-to-fine optical flow estimation



Gaussian pyramid of Image1

Gaussian pyramid of Image2

Optimization of Translational Movement

- Up to now: discrete search, usually pixel-wise
- how about subpixel values for $\boldsymbol{u}$?

- Assume, we found as optimum some $\boldsymbol{u}$
- Starting from there, we want to make small steps $\boldsymbol{\Delta u}$ to further optimize $\boldsymbol{u}$ for some pixel x_i:

$$I_1(\boldsymbol{x_i} + \boldsymbol{u} + \boldsymbol{\Delta u}) \approx I_1(\boldsymbol{x_i} + \boldsymbol{u}) + J_1(\boldsymbol{x_i} + \boldsymbol{u})\boldsymbol{\Delta}\mathbf{u}$$

using the first order Taylor expansion of I_1 with

$$\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) = \nabla I_1(\boldsymbol{x}_i + \boldsymbol{u}) = (\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y})(\boldsymbol{x}_i + \boldsymbol{u})$$

Optimization of Translational Movement

- $J_1(x)$ is the Jacobian of Image 1 in x
- i.e. the row vector of the derivatives of I_1 in x and y
- use central differences or forward or backward differences

- $J_1(x)\Delta\mathrm{x}$: how much does the color in I_1 change, if we move from x in direction Δx ?
- Obviously, this works well if we move by less than a pixel → we move within a linear interpolation
- For larger steps, it only works if the image is smooth

Optimization of Translational Movement

- Method of Lucas-Kanade (1981):

$$\begin{aligned} E_{\mathrm{LK-SSD}}(\boldsymbol{u} + \Delta\boldsymbol{u}) &= \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u} + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) + \boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 \\ &= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2, \end{aligned}$$

- with $e_i = I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)$

Optimization of Translational Movement

- set $A = \begin{pmatrix} J_1(x_1 + u) \\ \vdots \\ J_1(x_n + u) \end{pmatrix} \in \mathbb{R}^{n \times 2}$ and $b = \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} \in \mathbb{R}^n$
- *b* is called *residuum*
- these matrices and vectors are huge (one row per pixel)
- usually, we won't do this for an entire image, but for smaller patches (segmented foreground object, sub-patch → see later)

- Then we get as error:

$$E(\Delta u) = ||A\Delta u - b||^2$$

Optimization of Translational Movement

$$(a-b)^2 = a^2 - 2ab + b^2$$

- We want to minimize this error:

$$E(\Delta u) = ||A\Delta u - b||^2$$

- This is a typical **non-linear least squares problem**
- We write:

$$E(\Delta u) = \Delta u^T \left(A^T A\right) \Delta \mathrm{u} - 2\Delta u^T \left(A^T b\right) + b^T b$$

- and find the minimum by setting the derivative to zero:

$$2A^T A\Delta u - 2A^T b = 0$$

$$\underbrace{A^T A}_{\text{2x2 matrix}}\Delta u = \underbrace{A^T b}_{\text{2D vector}}$$

- and solve this equation for Δu

solution similar as for the camera matrix estimation

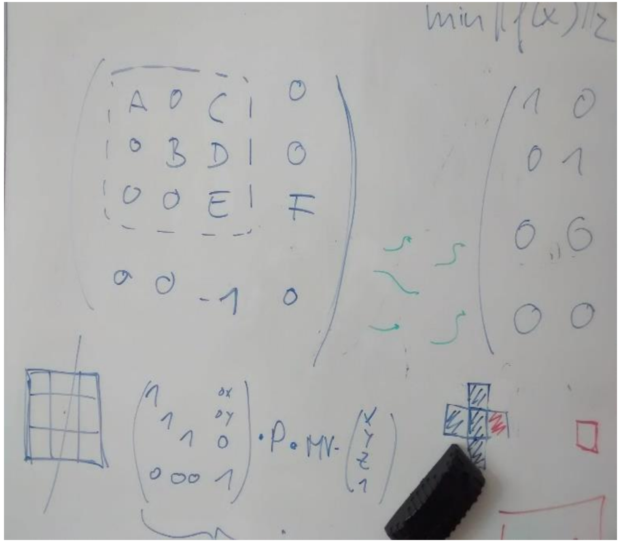
Optimization of Translational Movement

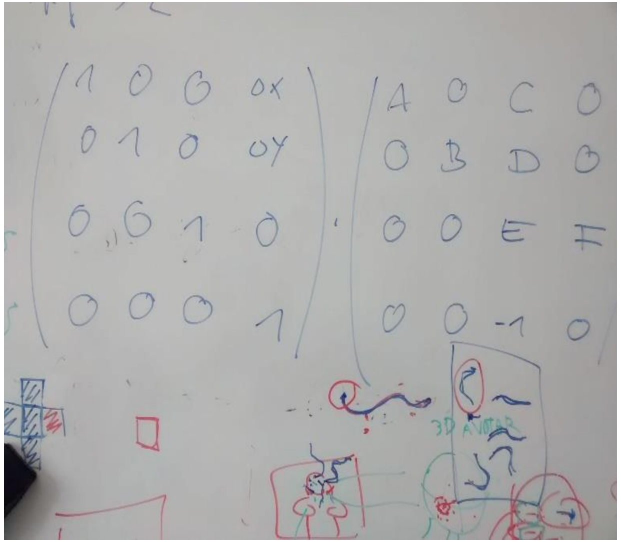
- We iterate this, and to have better stability, we only proceed a fraction α in direction of $\boldsymbol{\Delta u}$:

```
// given: starting value u
repeat
    set A = Jacobian of I1 at u
    set b = residual vector
    solve A^T*A*delta_u = A^T*b for delta_u
    u += alpha * delta_u
until convergence
```

Parametric Motion

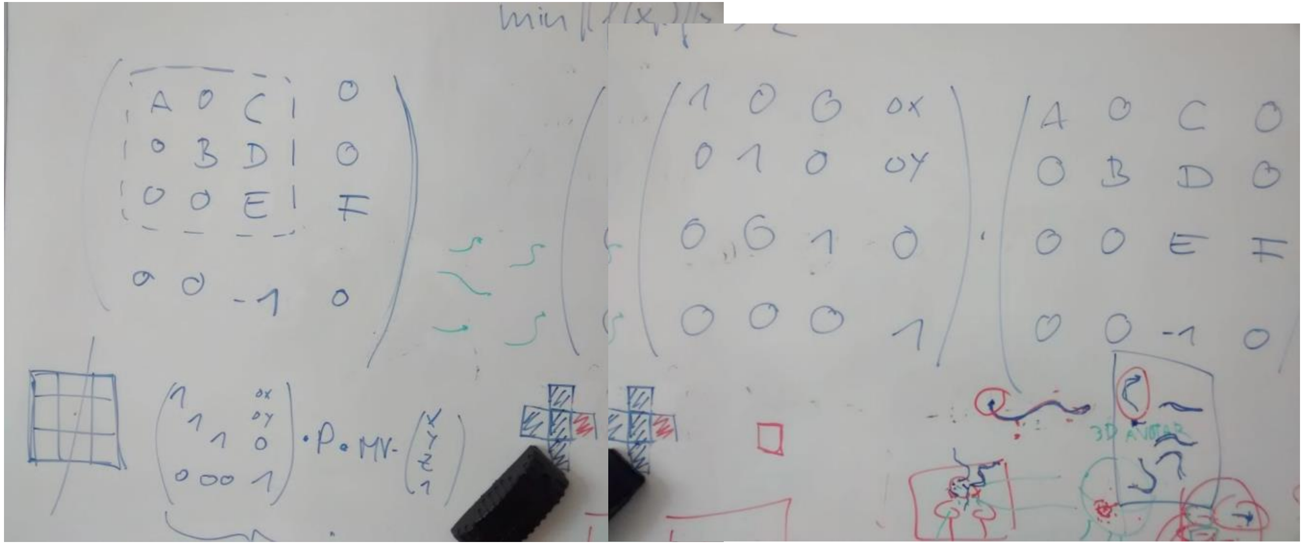
- Simple translation model maybe okay to stitch scanned documents → rotation disallowed





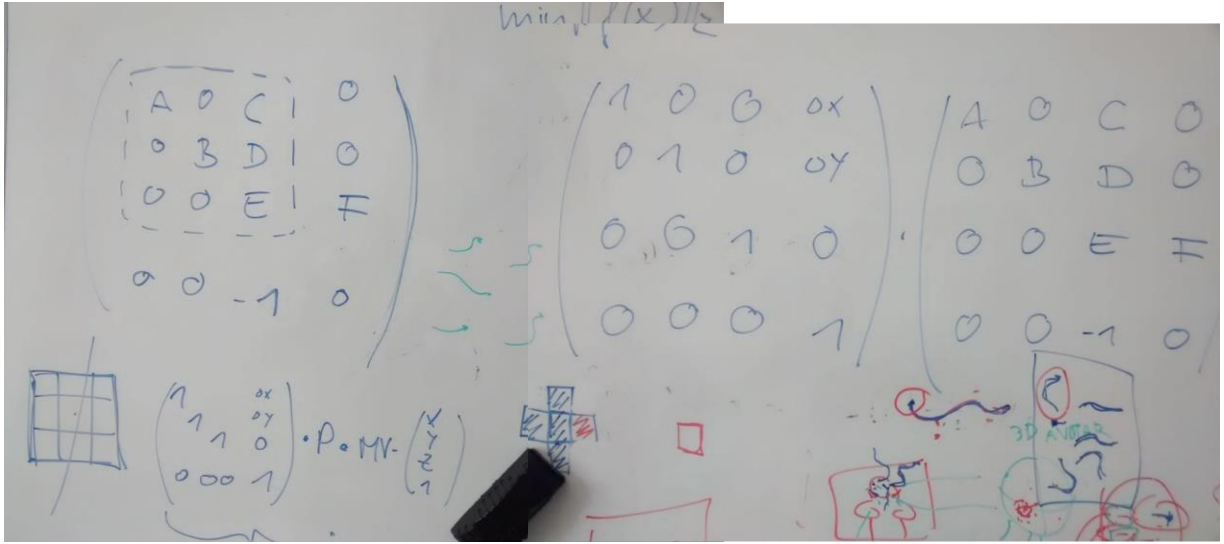
Parametric Motion

- Simple translation model maybe okay to stitch scanned documents → rotation disallowed



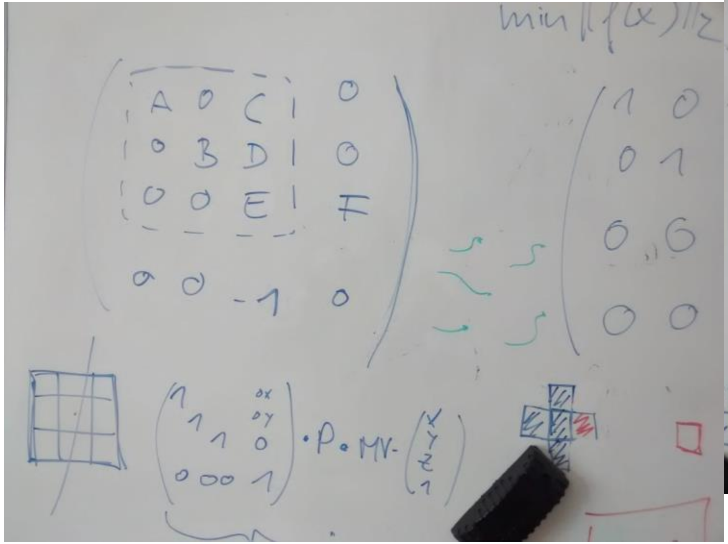
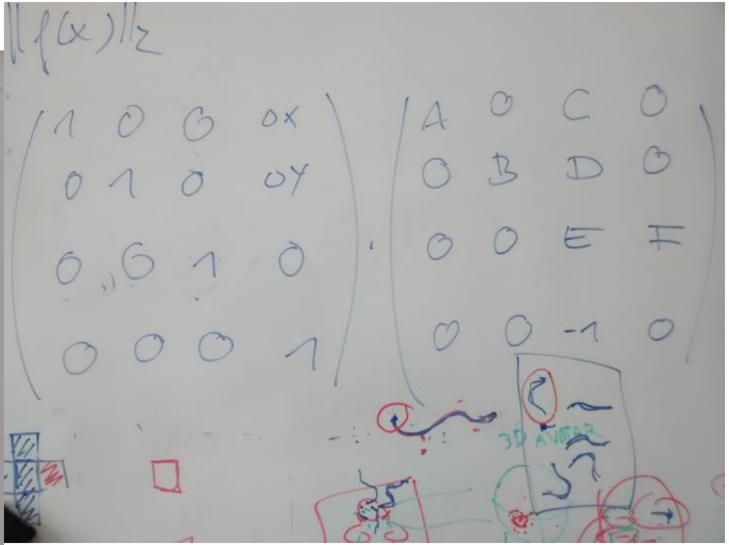
Parametric Motion

- Simple translation model maybe okay to stitch scanned documents → rotation disallowed



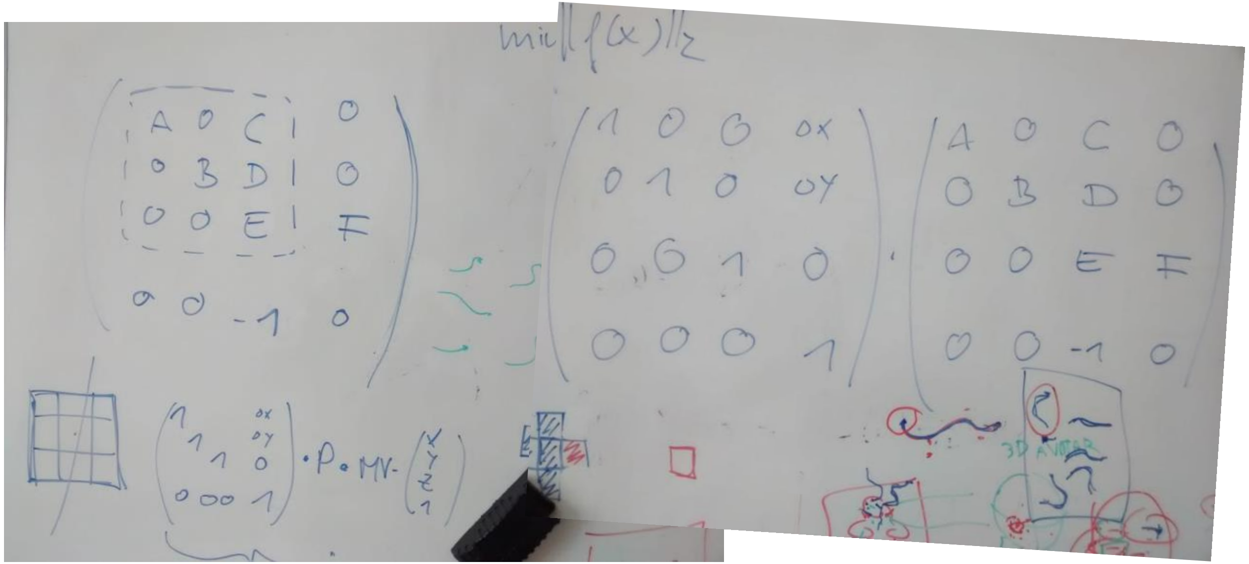
Parametric Motion

- Rotation can be necessary

Parametric Motion

- Rotation can be necessary



Parametric Motion

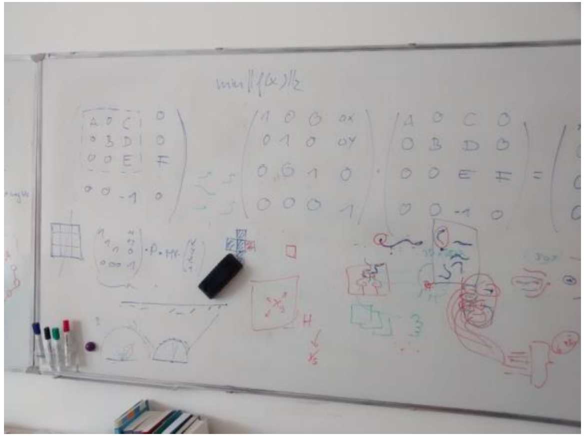
- $\rightarrow$ Search for translation and rotation
- Parameters: translation u and rotation ϕ
- $E_{SSD}(u, \Phi) = \sum_i \left(I_1(R(\Phi)x_i + u) - I_0(x_i)\right)^2$
  where $R(\Phi)$ is the rotation matrix for Φ

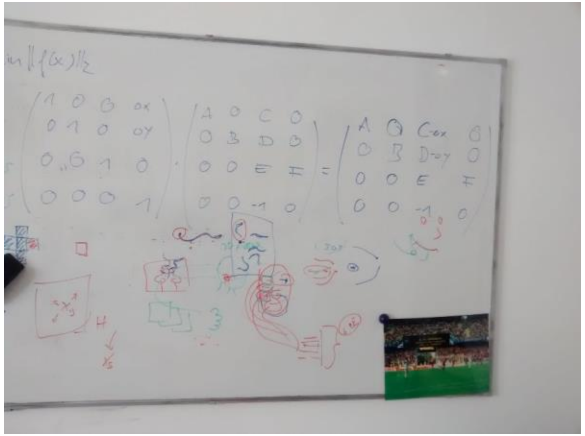
- New problem:

$$\arg\min E_{SSD}(u, \Phi)$$

Parametric Motion

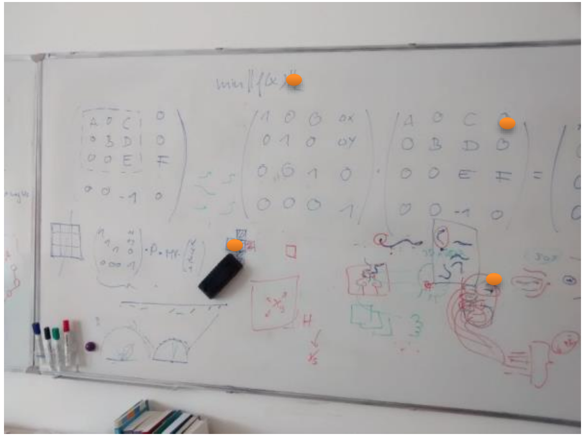
- Even more complicated: camera rotation
  → image planes not parallel
  - Projective mapping needed: 8 degrees of freedom
  - Needed for panorama stitching !

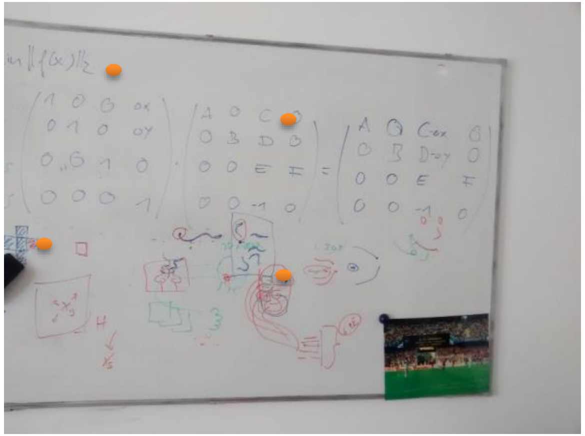




Parametric Motion

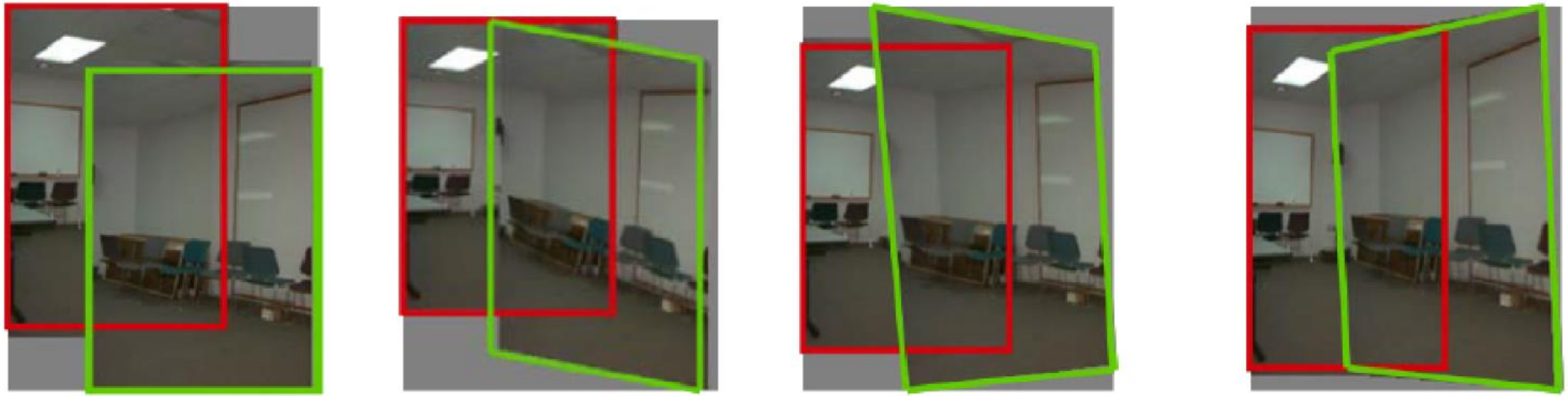
- Even more complicated: camera rotation
  - projective mapping: defined by mapping of four image points
    → 8 degrees of freedom





Parametric Motion

- Different motion models possible, e.g.:



(a) translation [2 dof] (b) affine [6 dof] (c) perspective [8 dof] (d) 3D rotation [3+ dof]

Parametric Motion

- We describe parametric motion with a parameter vector p
- Motion is then $x \to x'(x, p)$
- Lucas-Kanade with parametric motion:

$$\begin{aligned} E_{\mathrm{LK-PM}}(\boldsymbol{p}+\Delta\boldsymbol{p}) &= \sum_i [I_1(\boldsymbol{x}'(\boldsymbol{x}_i;\boldsymbol{p}+\Delta\boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i [I_1(\boldsymbol{x}'_i) + \boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} - I_0(\boldsymbol{x}_i)]^2 \\ &= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} + e_i]^2, \end{aligned}$$

- where J_1 is now with respect to parameter vector p:

$$J_1(x') = \frac{\partial I_1(x')}{\partial x'} \cdot \frac{\partial x'(x)}{\partial p}$$

gradient from image

gradient from motion model

Parametric Motion

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	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	$\Downarrow$

Parametric Motion

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \text{ and } y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}.$$

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}$$

General Optical Flow

- Optical Flow: one motion vector per pixel
- General approach:

$$E_{\mathrm{SSD-OF}}(\{\boldsymbol{u}_i\}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}_i) - I_0(\boldsymbol{x}_i)]^2.$$

motion of pixel i

- cannot work directly:
  - each u_i is 2D, so we have $2n$ unknowns (n is the number of pixels), but only n equations
- obvious solution:
  - for each pixel in I_0, find some pixel in I_1 with same color → u_i
  - 1 million pixels, 256 colors → many, many solutions

Patch-based Optical Flow

- Patch-based approach:
  - for each pixel (or a coarser grid on the image) compute translational flow on a local neighborhood to determine the pixel's flow
- Algorithm as before, but on smaller patches
- Also works on image pyramid
- runs into problems for disocclusions
  (newly visible region will always generate error)
  → problematic near silhouettes
  → see later: layer-based flow

Patch-based Optical Flow

- Observation: solution not always obvious

- low textured region: optimal translation unclear

Patch-based Optical Flow

- Observation: solution not always obvious

- at edges:
  translation along
  edge leads to
  similar error

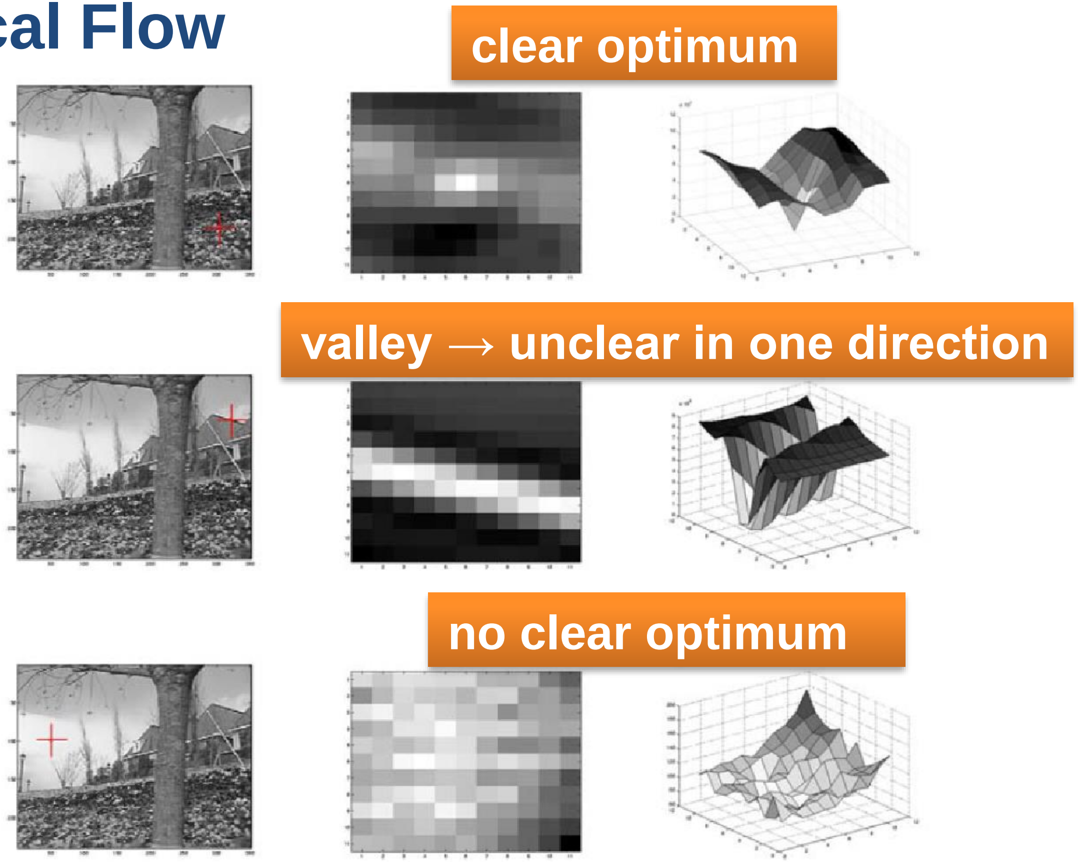
Patch-based Optical Flow

- Observation: solution not always obvious

- highly textured region: works best
- Repetitive structures can be tricky

Patch-based Optical Flow

- SSD surfaces: at the three image locations, offset a small neighborhood and compute SSD error



Regularization-based Optical Flow

- General optical flow underconstrained:

$$E_{SSD-OF}(\{u_i\}) = \sum_i [I_1(x_i + u_i) - I_0(x_i)]^2$$

- Regularization-based flow
  - $\boldsymbol{u_i}$ should not be chosen arbitrarily
  - instead: neighboring $\boldsymbol{u_i}$ should be similar
  - add regularization term that penalizes dissimilar $\boldsymbol{u_i}$:

$$E_{SSD-OF}(\{u_i\}) = \sum_i [I_1(x_i + u_i) - I_0(x_i)]^2 + \alpha^2 \|\nabla u_i\|^2$$

FlowNet

- Estimate optical Flow in a data driven way

FlowNet: Learning Optical Flow with Convolutional Networks

Philipp Fischer[*,‡], Alexey Dosovitskiy[‡], Eddy Ilg[‡], Philip Häusser, Caner Hazırbaş, Vladimir Golkov[*]
University of Freiburg — Technical University of Munich
{fischer,dosovits,ilg}@cs.uni-freiburg.de, {haeusser,hazirbas,golkov}@cs.tum.edu

Patrick van der Smagt, Technical University of Munich, smagt@brml.org
Daniel Cremers, Technical University of Munich, cremers@tum.de
Thomas Brox, University of Freiburg, brox@cs.uni-freiburg.de

2015

FlowNet

- Estimate optical Flow in a data driven way

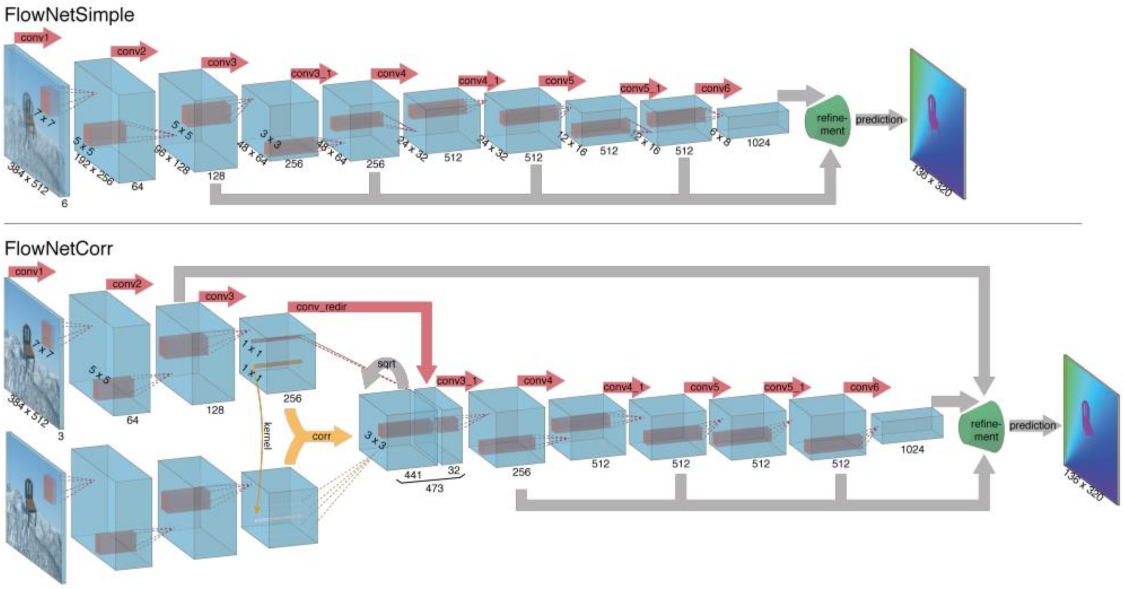


Figure 2. The two network architectures: FlowNetSimple (top) and FlowNetCorr (bottom).

FlowNet

- Estimate optical Flow in a data driven way
- Synthetic training data
  - Ground truth flow
  - Real motion, not apparent motion

FlowNet 2.0

FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks

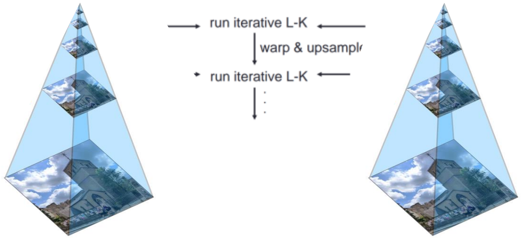
Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, Thomas Brox
University of Freiburg, Germany
{ilg,mayern,saikiat,keuper,dosovits,brox}@cs.uni-freiburg.de

] 6 Dec 2016

Abstract

The FlowNet demonstrated that optical flow estimation can be cast as a learning problem. However, the state of the art with regard to the quality of the flow has still been defined by traditional methods. Particularly on small displacements and real-world data, FlowNet cannot compete with variational methods. In this paper, we advance the concept of end-to-end learning of optical flow and make it

FlowNet 2.0



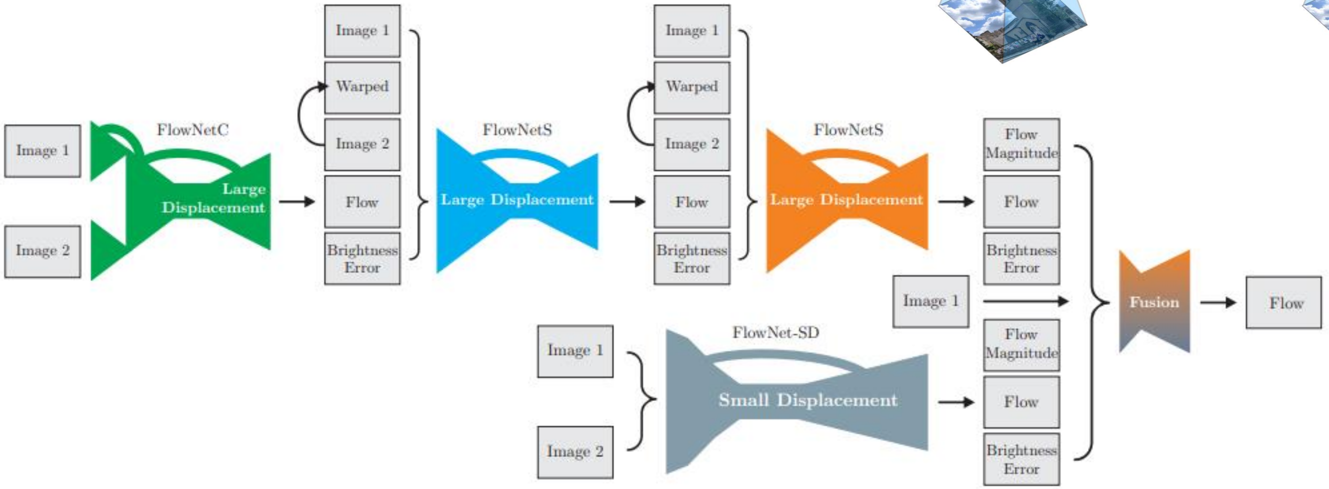


Figure 2. Schematic view of complete architecture: To compute large displacement optical flow we combine multiple FlowNets. Braces indicate concatenation of inputs. *Brightness Error* is the difference between the first image and the second image warped with the previously estimated flow. To optimally deal with small displacements, we introduce smaller strides in the beginning and convolutions between upconvolutions into the FlowNetS architecture. Finally we apply a small fusion network to provide the final estimate.

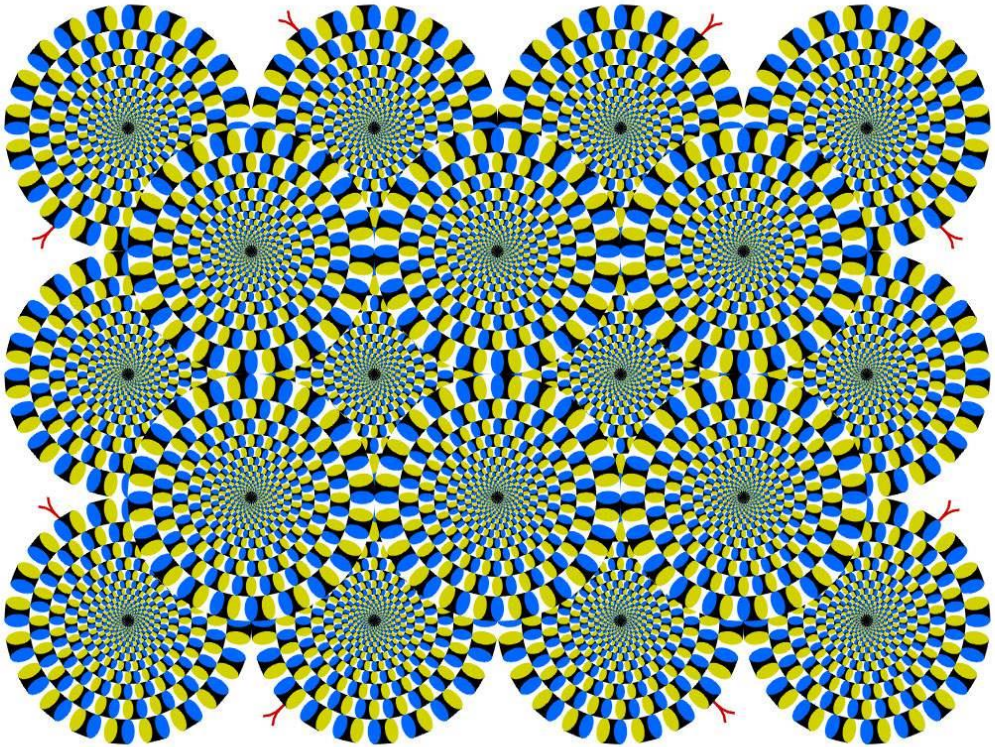
Example application

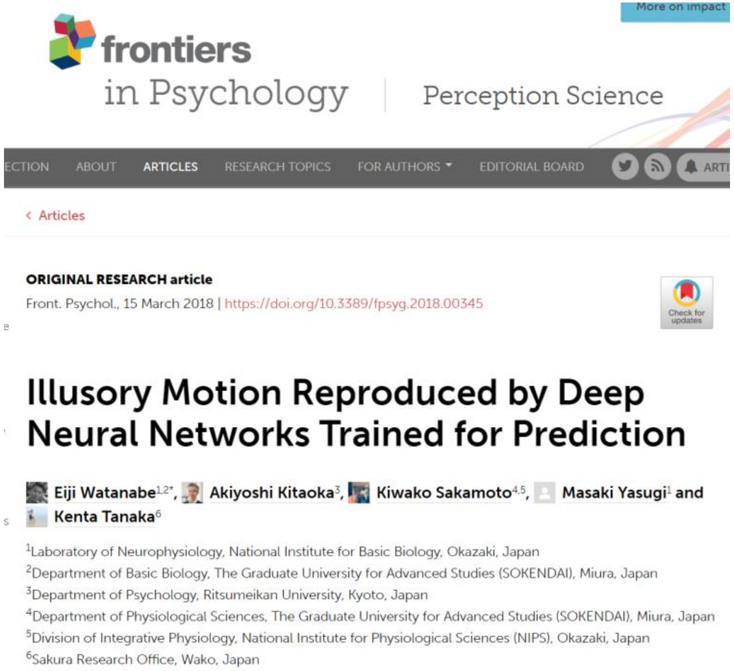
- https://www.youtube.com/watch?v=iOcXdGZUvSo

Example application

- https://www.youtube.com/watch?v=wZcBLc4ifuQ

Rotating Snakes *Akiyoshi Kitaoka*





More on impact

frontiers in Psychology | Perception Science

ECTION ABOUT ARTICLES RESEARCH TOPICS FOR AUTHORS EDITORIAL BOARD ARTI

< Articles

ORIGINAL RESEARCH article

Front. Psychol., 15 March 2018 | https://doi.org/10.3389/fpsyg.2018.00345

Check for updates

Illusory Motion Reproduced by Deep Neural Networks Trained for Prediction

Eiji Watanabe[1,2*], Akiyoshi Kitaoka[3], Kiwako Sakamoto[4,5], Masaki Yasugi[1] and Kenta Tanaka[6]

[1]Laboratory of Neurophysiology, National Institute for Basic Biology, Okazaki, Japan
[2]Department of Basic Biology, The Graduate University for Advanced Studies (SOKENDAI), Miura, Japan
[3]Department of Psychology, Ritsumeikan University, Kyoto, Japan
[4]Department of Physiological Sciences, The Graduate University for Advanced Studies (SOKENDAI), Miura, Japan
[5]Division of Integrative Physiology, National Institute for Physiological Sciences (NIPS), Okazaki, Japan
[6]Sakura Research Office, Wako, Japan