Visual Navigation for Flying Robots

Visual Motion Estimation

Dr. Jürgen Sturm

Organization: Exam

- Registration deadline: June 30
- Course ends: July 19
- Examination dates: t.b.a. (mid August)
  - Oral team exam
  - Sign up for a time slot starting from Mid July
  - List will be placed on blackboard in front of our secretary

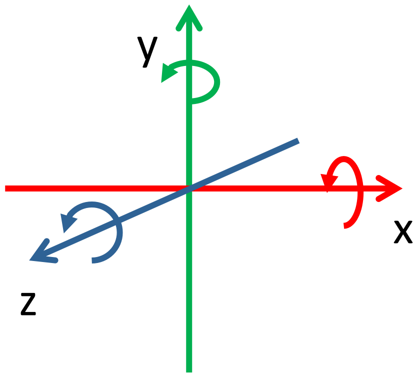
Motivation

Visual Motion Estimation

- Quick geometry recap
- Image filters
- 2D image alignment
- Corner detectors
- Kanade-Lucas-Tomasi tracker
- 3D motion estimation

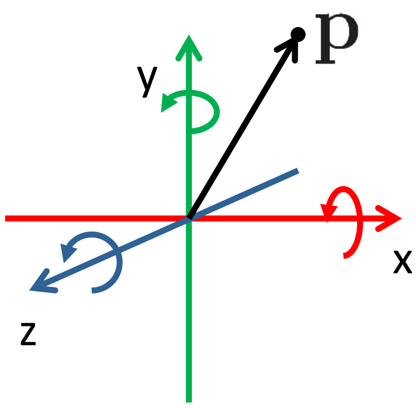
Angular and linear velocities

- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Linear and angular velocity together form a twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top, \boldsymbol{\omega}^\top)^\top$



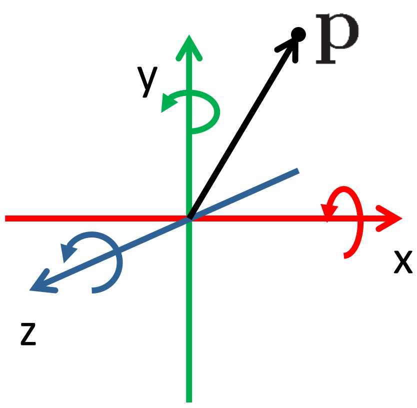
Angular and linear velocities

- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top, \boldsymbol{\omega}^\top)^\top$



Angular and linear velocities

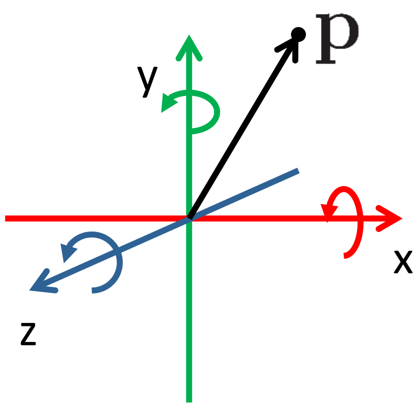
- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top, \boldsymbol{\omega}^\top)^\top$
- What is the velocity $\dot{\mathbf{p}}$ at point $\mathbf{p}$?



Angular and linear velocities

- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top, \boldsymbol{\omega}^\top)^\top$
- What is the velocity $\dot{\mathbf{p}}$ at point $\mathbf{p}$?

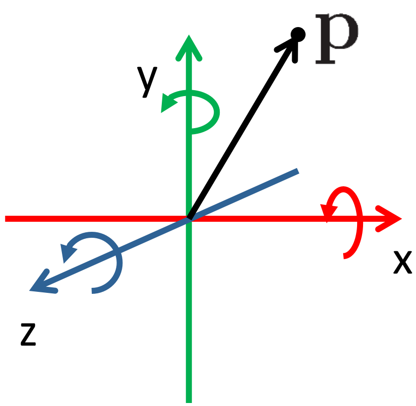
$$\begin{aligned}\mathbf{p}(t) &= R(t)\mathbf{p}(0) + \mathbf{t}(t) \\ &= \exp([\boldsymbol{\omega}]_\times t)\mathbf{p}(0) + \boldsymbol{v}t\end{aligned}$$



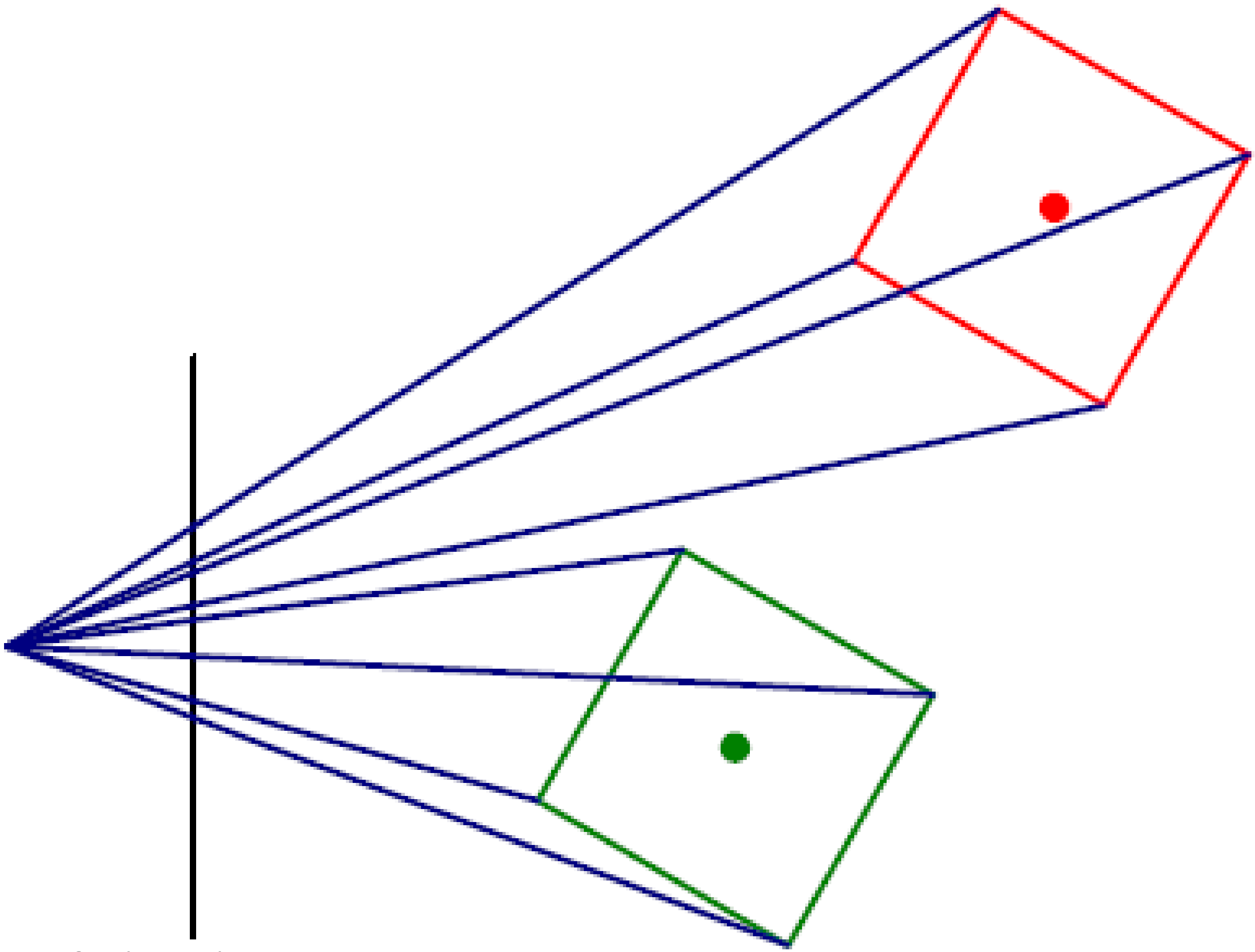
Angular and linear velocities

- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top, \boldsymbol{\omega}^\top)^\top$
- What is the velocity $\dot{\mathbf{p}}$ at point $\mathbf{p}$?

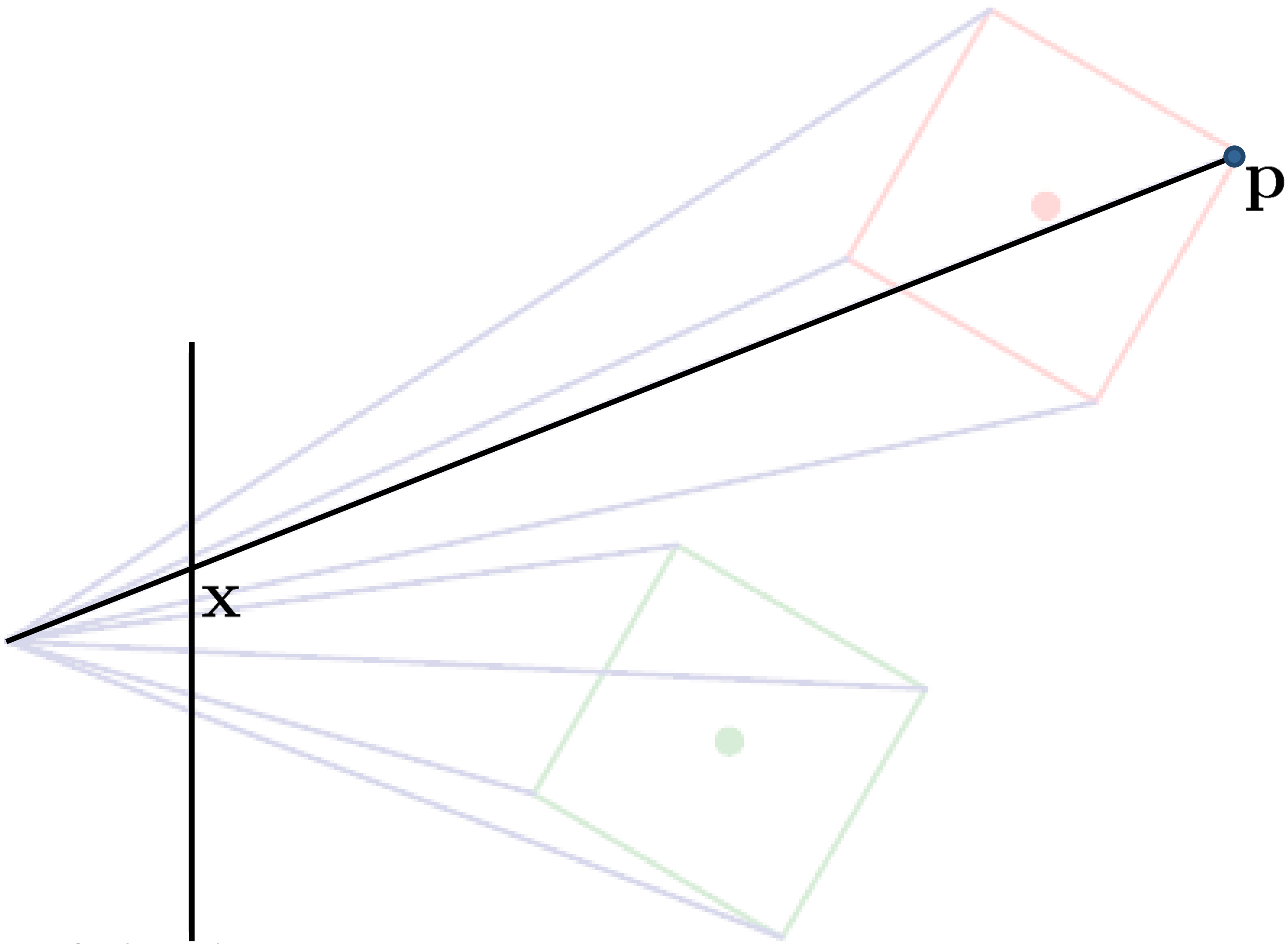
$$\mathbf{p}(t) = R(t)\mathbf{p}(0) + \mathbf{t}(t)$$
$$= \exp([\boldsymbol{\omega}]_\times t)\mathbf{p}(0) + \boldsymbol{v}t$$
$$\Rightarrow \dot{\mathbf{p}}(t) = \exp([\boldsymbol{\omega}]_\times t)[\boldsymbol{\omega}]_\times \mathbf{p}(0) + \boldsymbol{v}$$
$$\dot{\mathbf{p}}(0) = [\boldsymbol{\omega}]_\times \mathbf{p}(0) + \boldsymbol{v}$$



Recap: Perspective Projection



Recap: Perspective Projection



3D to 2D Perspective Projection

- 3D point $\mathbf{p}$ (in the camera frame)
- 2D point $\mathbf{x}$ (on the image plane)
- Pin-hole camera model

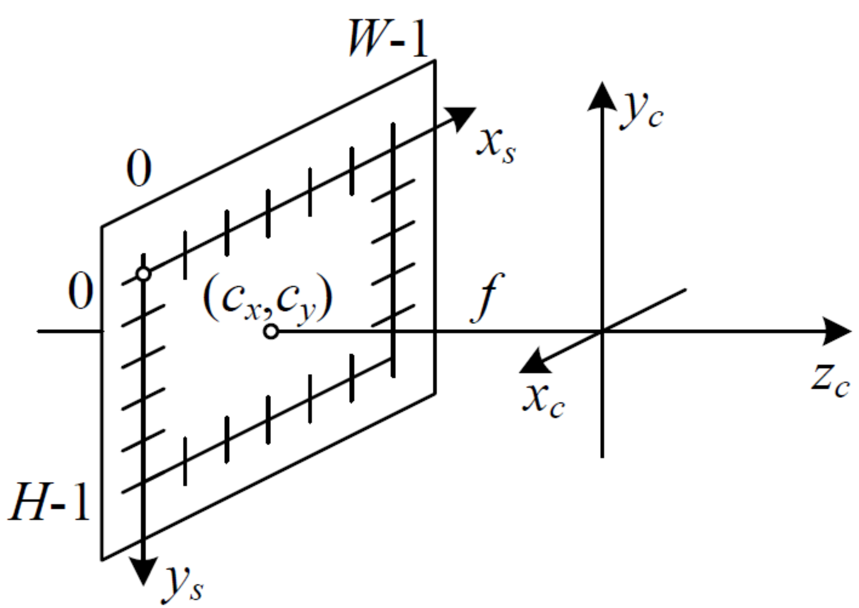
$$\tilde{\mathbf{x}} = \lambda \bar{\mathbf{x}} = \mathbf{p}$$

- Remember, $\tilde{\mathbf{x}}$ is homogeneous, need to normalize

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \end{pmatrix}$$

Camera Intrinsics

- So far, 2D point is given in meters on image plane
- But: we want 2D point be measured in pixels (as the sensor does)



Camera Intrinsics

- Need to apply some scaling/offset

$$\tilde{\mathbf{x}} = \underbrace{\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsics } K} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \tilde{\mathbf{p}}$$

- Focal length f_x, f_y
- Camera center c_x, c_y
- Skew s

Image Plane

- Pixel coordinates $\mathbf{x} \in \Omega$
- Image plane $\Omega \subset \mathbb{R}^2$

- Example:
  - Discrete case $\mathbf{x} \in [0, W) \times [0, H) \subset \mathbb{N}_0^2$ (default in this course)
  - Continuous case $\mathbf{x} \in [0, 1] \times [0, 1] \subset \mathbb{R}^2$

Image Functions

- We can think of an image as a function $f : \Omega \mapsto \mathbb{R}$
- $f(\mathbf{x})$ gives the intensity at position $\mathbf{x}$
- Color images are vector-valued functions

$$f(\mathbf{x}) = \begin{pmatrix} r(\mathbf{x}) \\ g(\mathbf{x}) \\ b(\mathbf{x}) \end{pmatrix}$$

Image Functions

- Realistically, the image function is only defined on a rectangle and has finite range

$$f : [0, W-1] \times [0, H-1] \mapsto [0,1]$$

- Image can be represented as a matrix
- Alternative notations

$$F_{ij}, f(i,j), f(x,y), f(\mathbf{x}), \ldots$$

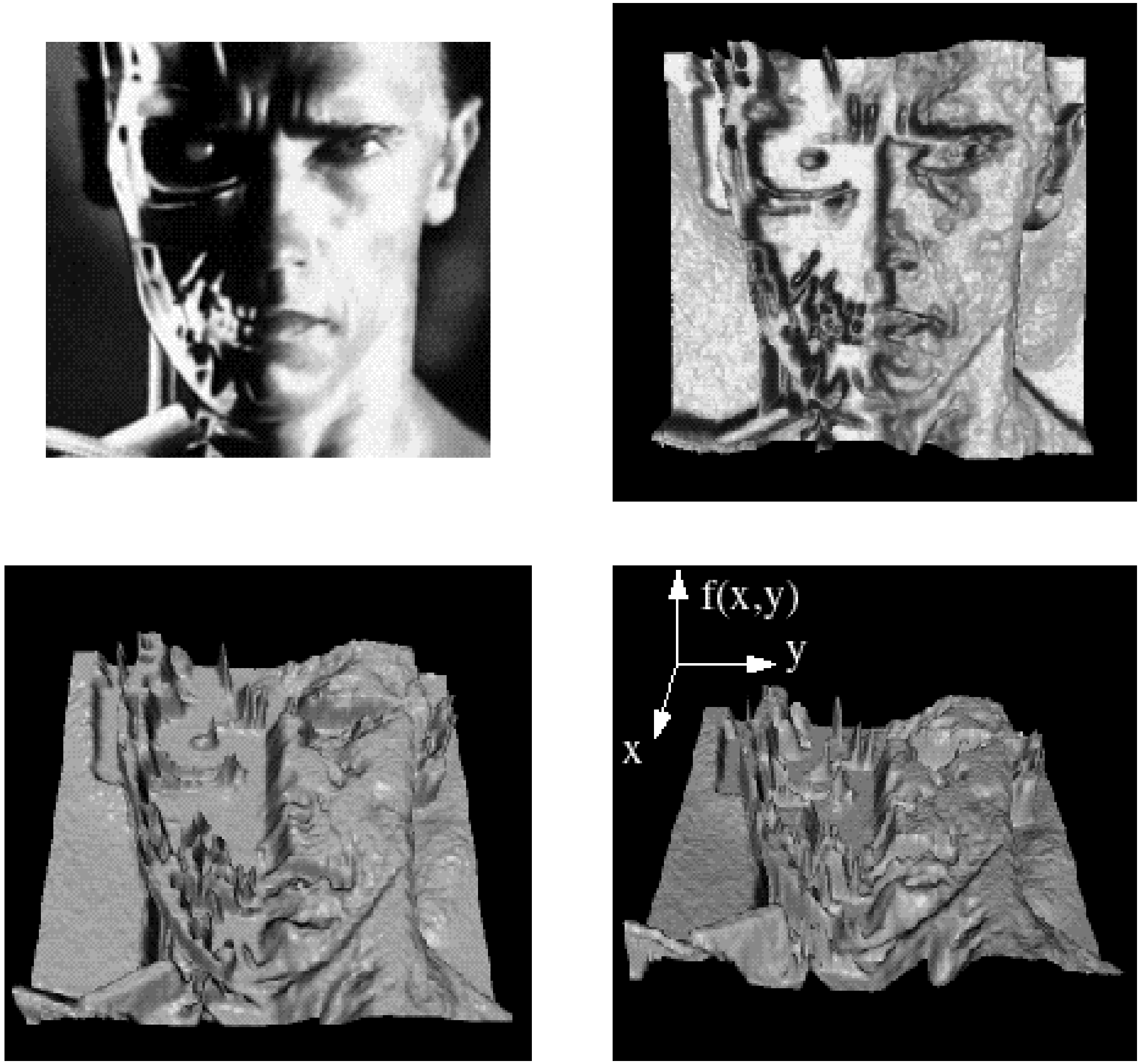
often (row,column)

often (column,row)

j (→), i (↓)

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

Example



Digital Images

- Light intensity is sampled by CCD/CMOS sensor on a regular grid
- Electric charge of each cell is quantized and gamma compressed (for historical reasons)

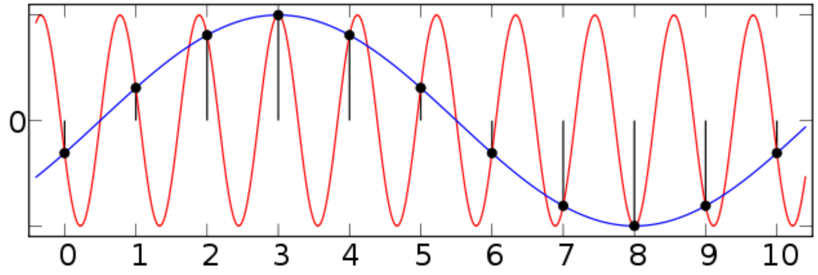
$$V = B^{\frac{1}{\gamma}} \text{ with } \gamma = 2.2$$

- CRTs / monitors do the inverse $B = V^{\gamma}$
- Almost all images are gamma compressed

→ Double brightness results only in a 37% higher intensity value (!)

Aliasing

- High frequencies in the scene and a small fill factor on the chip can lead to (visually) unpleasing effects
- Examples

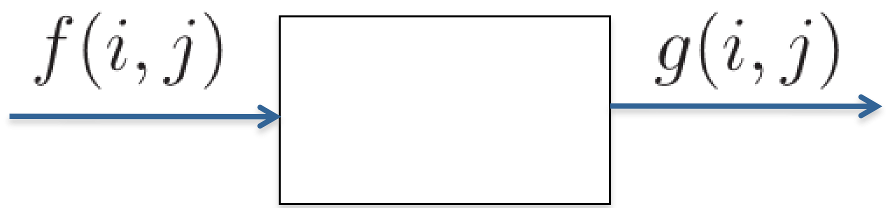


Rolling Shutter

- Most CMOS sensors have a rolling shutter
- Rows are read out sequentially
- Sensitive to camera and object motion
- Can we correct for this?

Image Filtering

- We want to remove unwanted sources of variation, and keep the information relevant for whatever task we need to solve



- Example tasks:
  de-noising, (de-)blurring, computing derivatives, edge detection, …

Linear Filtering

- Each output is a linear combination of all the input values

$$g(i,j) = \sum_{k,l} h(i,j,k,l) f(k,l)$$

- In matrix form

G = H F

Spatially Invariant Filtering

- We are often interested in spatially invariant operations

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

- Example

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

$*$

-1	2	-1
-1	2	-1
-1	2	-1

$=$?

Spatially Invariant Filtering

- We are often interested in spatially invariant operations

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

- Example

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

$*$

-1	2	-1
-1	2	-1
-1	2	-1

$=$

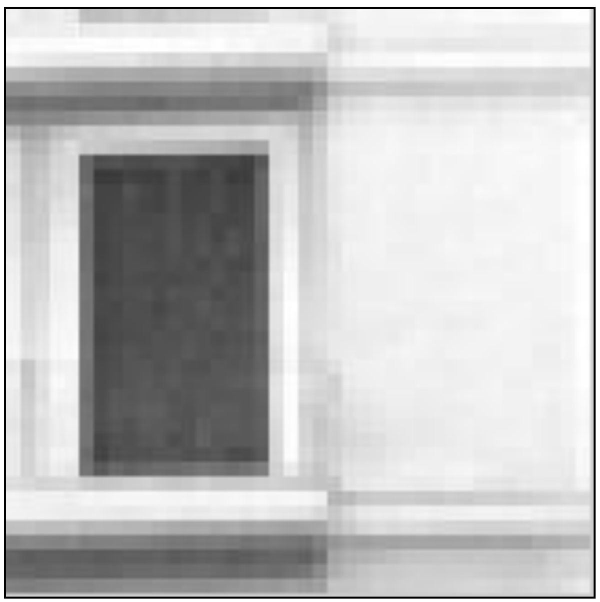
?	?	?	?	?	?	?	?
?	-5	9	-9	21	-12	10	?
?	-29	18	24	4	-7	5	?
?	-50	40	142	-88	-34	10	?
?	-41	41	264	-175	-71	0	?
?	-24	37	349	-224	-120	-10	?
?	-23	33	360	-217	-134	-23	?
?	?	?	?	?	?	?	?

Important Filters

- **Impulses**
- **Shifts**
- Blur
  - **Gaussian**
  - Bilateral filter
  - Motion blur
- Edges
  - **Finite difference filter**
  - Derivative filter
  - Oriented filters
  - Gabor filter
- …

Impulse

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



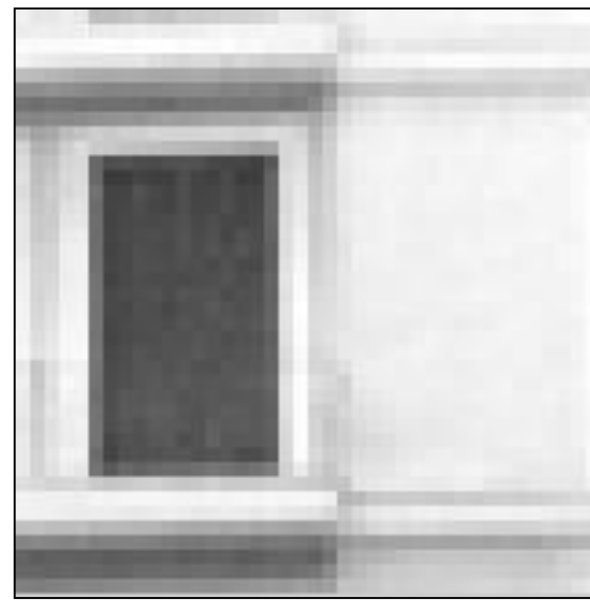
$f(i,j)$

$*$

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

$h(i,j)$

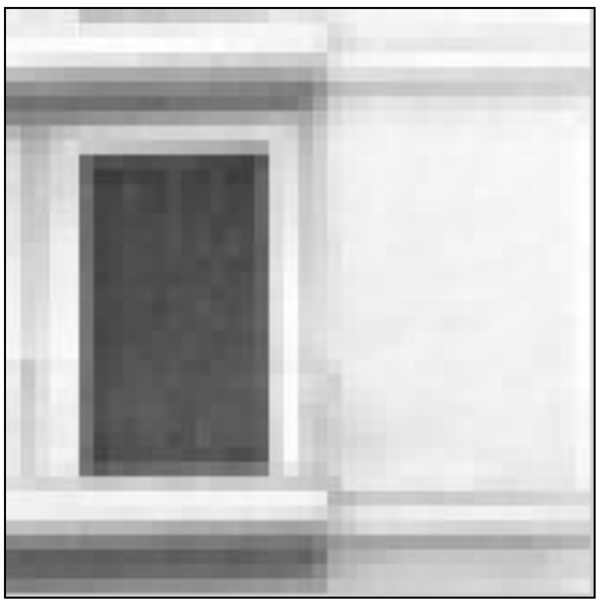
$=$



$g(i,j)$

Image shift (translation)

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



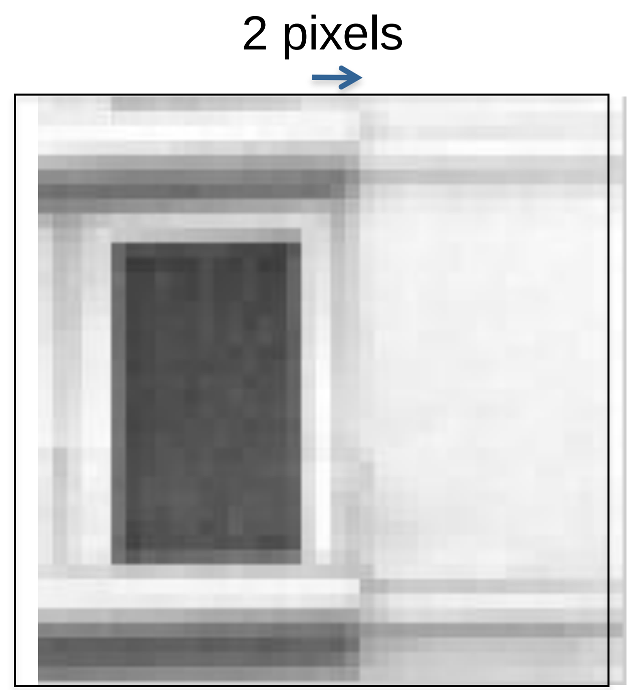
$f(i,j)$

$*$

0	0	0	0	0
0	0	0	0	0
1	0	0	0	0
0	0	0	0	0
0	0	0	0	0

$h(i,j)$

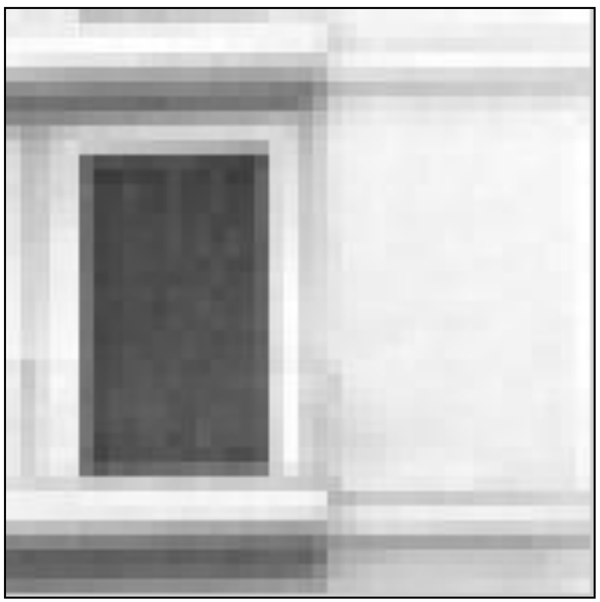
$=$



$g(i,j)$

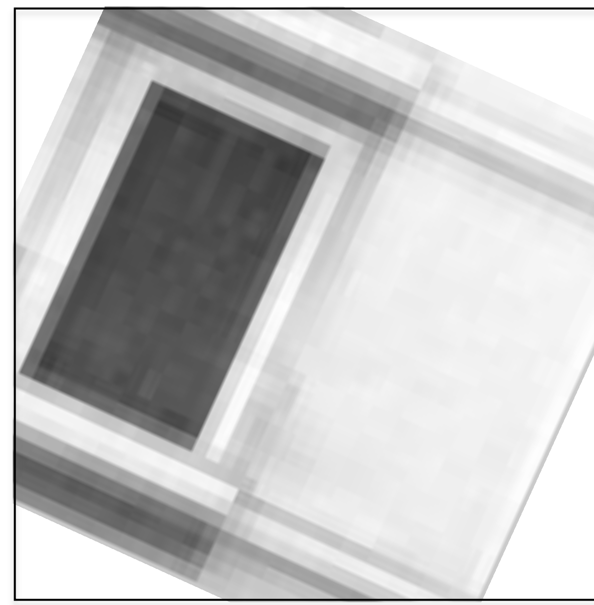
Image rotation

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



$f(i,j)$

$*$? $h(i,j)$ $=$

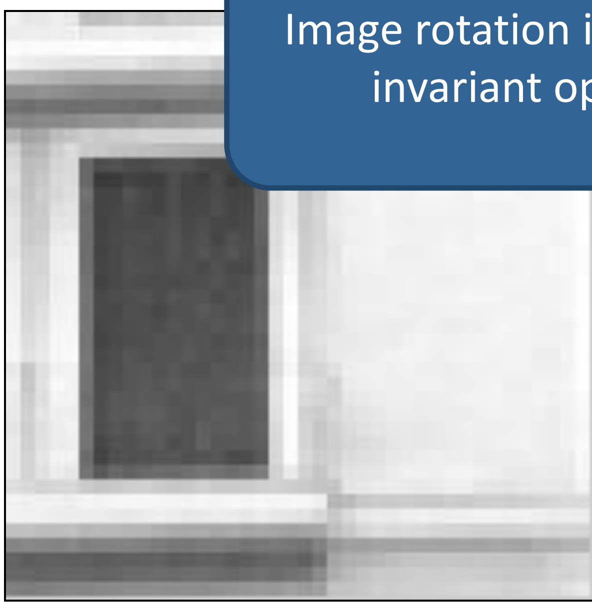


$g(i,j)$

Image rotation

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

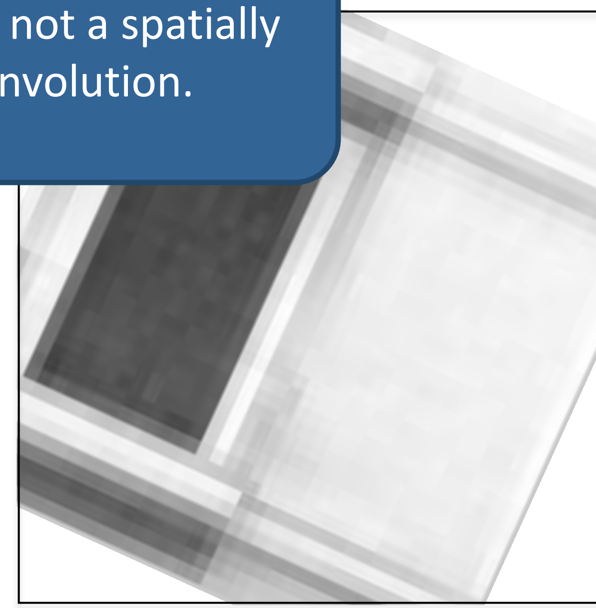
Image rotation is a linear operator (why?), but not a spatially invariant operation (why?). There is no convolution.



$f(i,j)$

$*$? $=$

$h(i,j)$

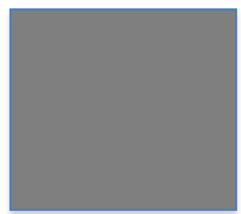


$g(i,j)$

Rectangular Filter

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

$*$



$=$

$f(i,j)$ $h(i,j)$ $g(i,j)$

Rectangular Filter

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

$f(i,j)$

$*$

$h(i,j)$

$=$

$g(i,j)$

Rectangular Filter

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

$f(i,j)$

$*$

$h(i,j)$

$=$

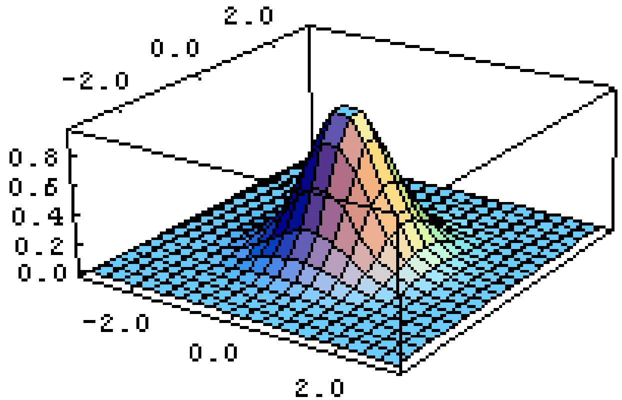
$g(i,j)$

Gaussian Blur

- Gaussian distribution

$$g_\sigma(i,i) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

- Example of resulting kernel



Gaussian Blur

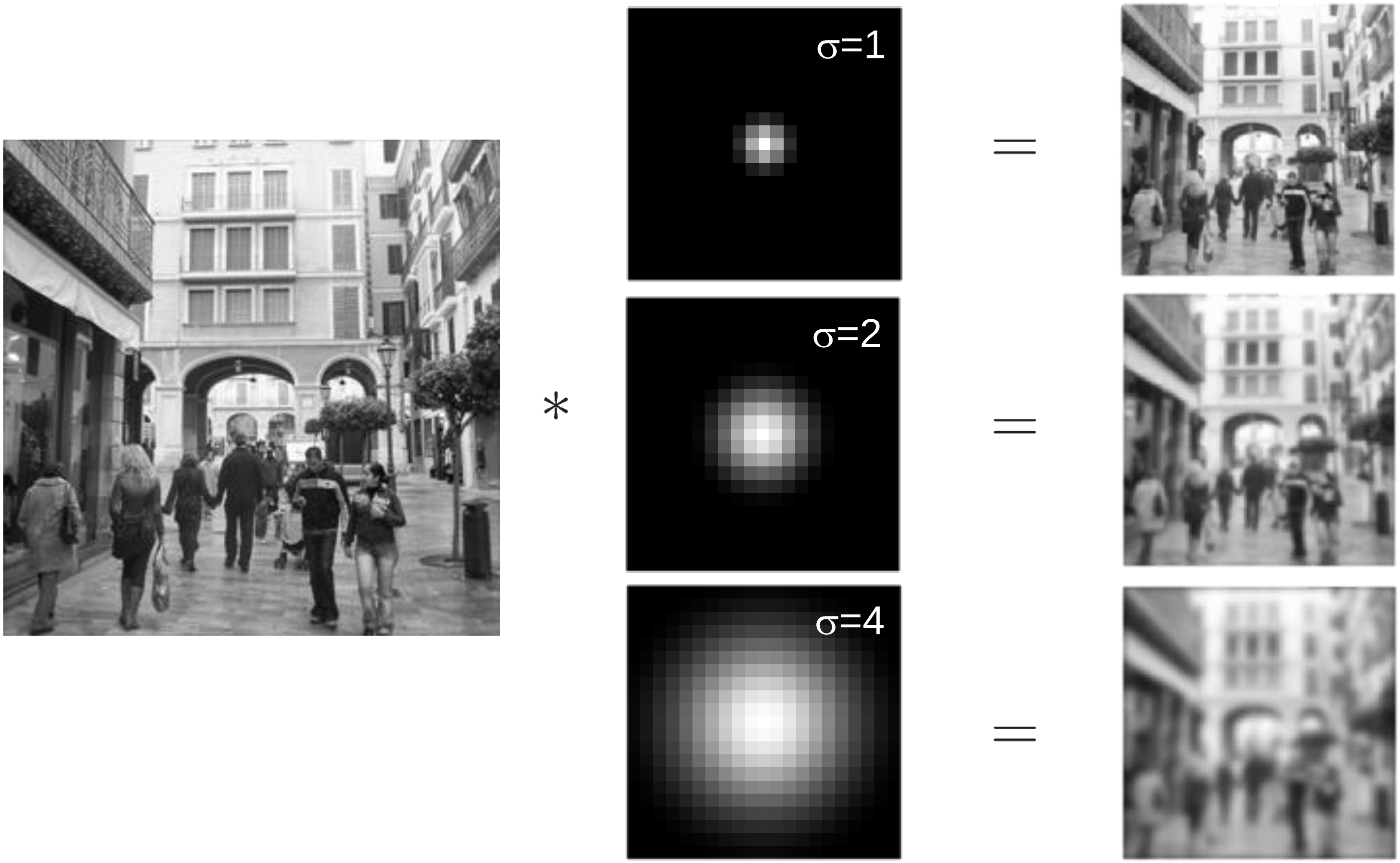
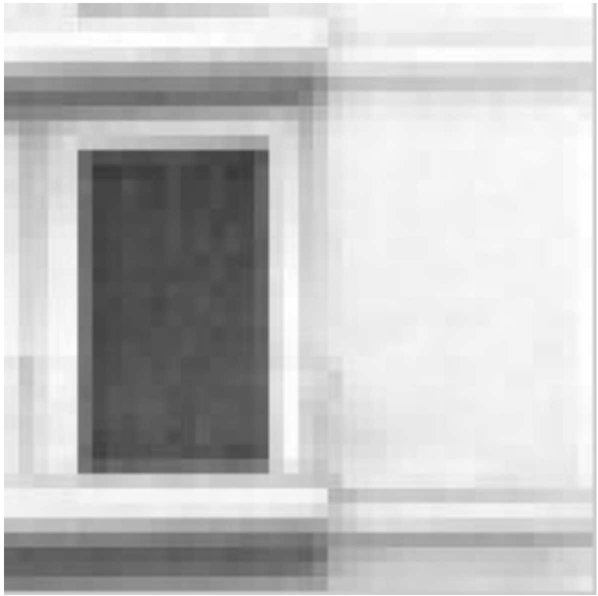


Image Gradient

- The image gradient $\nabla f = \left(\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \right)^\top$ points in the direction of increasing intensity (steepest ascend)



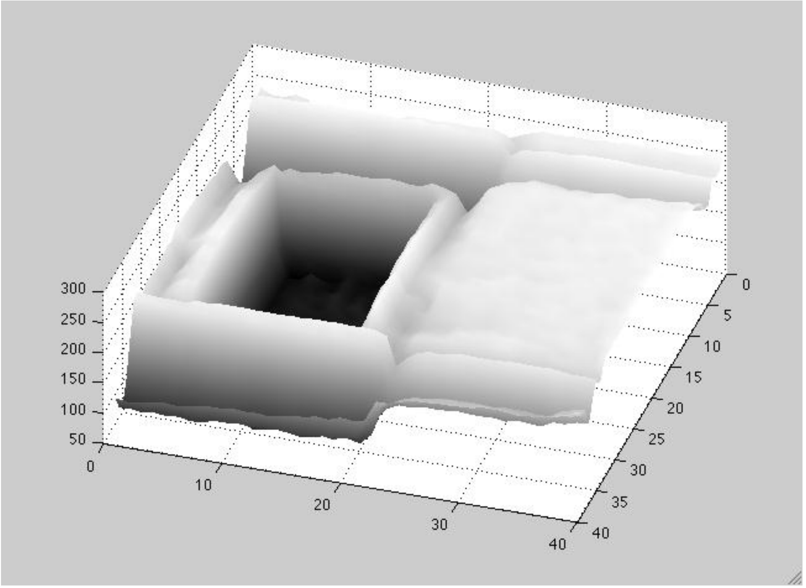
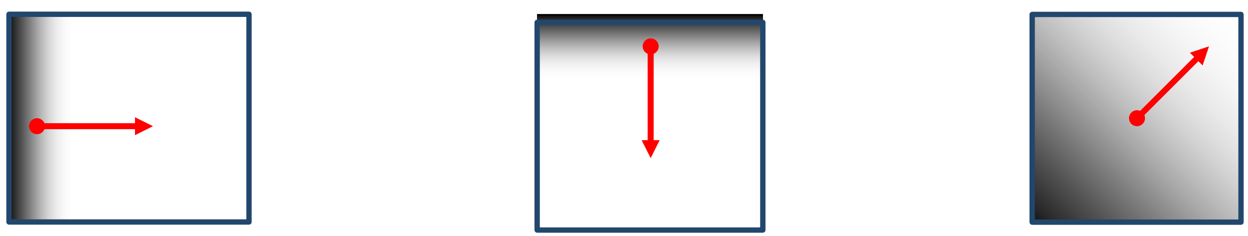


Image Gradient

- The image gradient $\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix}^\top$ points in the direction of increasing intensity (steepest ascend)



$$\nabla f = \left(\frac{\partial f}{\partial x}, 0 \right)^\top \quad \nabla f = \left(0, \frac{\partial f}{\partial y} \right)^\top \quad \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^\top$$

Image Gradient

- Gradient direction (related to edge orientation)

$$\theta = \operatorname{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$$

- Gradient magnitude (edge strength)

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Image Gradient

How can we differentiate a digital image $f(x, y)$?

- Option 1: Reconstruct a continuous image, then take gradient
- **Option 2: Take discrete derivative (finite difference filter)**
- Option 3: Convolve with derived Gaussian (derivative filter)

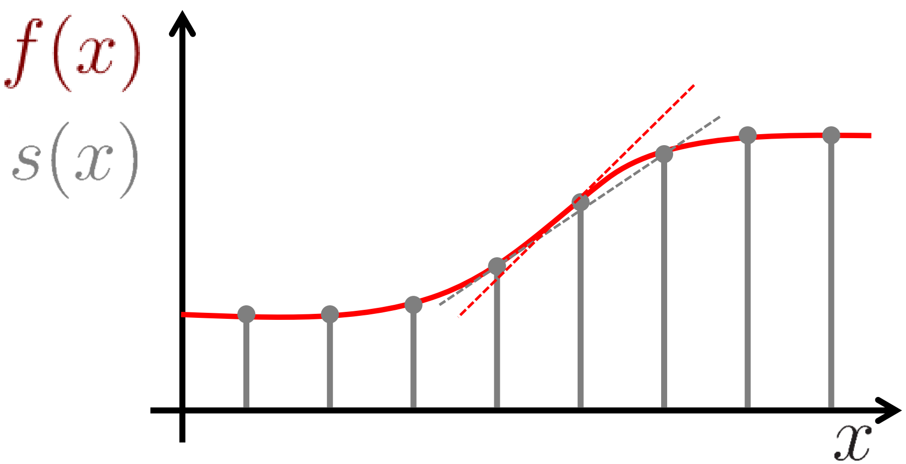
Finite difference

- First-order central difference

$$\frac{\partial f}{\partial x}(x,y) \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

- Corresponding convolution kernel:

-.5	0	.5



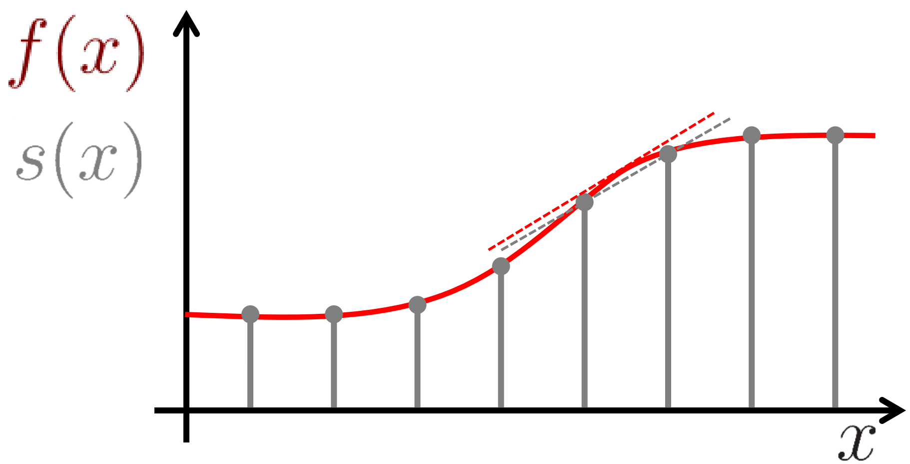
Finite difference

- First-order central difference (half pixel)

$$\frac{\partial f}{\partial x}(x, y) \approx f(x + 0.5, y) - f(x - 0.5, y)$$

- Corresponding convolution kernel:

-1	1



Second-order Derivative

- Differentiate again to get second-order central difference

$$\frac{\partial f(x)}{\partial x^2} \approx f(x+1) - 2f(x) + f(x-1)$$

Corresponding convolution kernel:

1	-2	1

Example

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

$f(i,j)$

$*$

-1	1

$h(i,j)$

$=$

$g(i,j)$

Example

$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

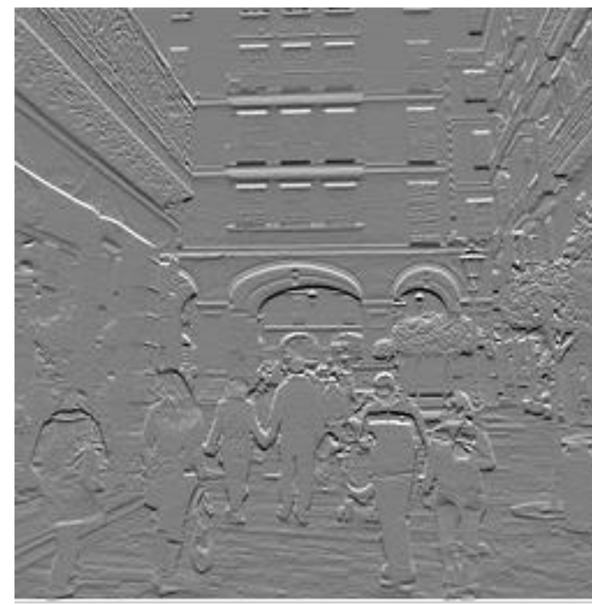
$f(i,j)$

$*$

-1
1

$h(i,j)$

$=$



$g(i,j)$

(Dense) Motion Estimation

- 2D motion

- 3D motion

Problem Statement

- **Given:** two camera images f_0, f_1
- **Goal:** estimate the camera motion $\mathbf{u}$

- For the moment, let's assume that the camera only moves in the xy-plane, i.e., $\mathbf{u} = (u\ v)^\top$
- Extension to 3D follows

General Idea

1. Define an error metric $E(\mathbf{u})$ that defines how well the two images match given a motion vector
2. Find the motion vector with the lowest error

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} E(\mathbf{u})$$

Error Metrics for Image Comparison

- Sum of Squared Differences (SSD)

$$E_{\mathrm{SSD}}(\mathbf{u}) = \sum_i (f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i))^2 = \sum_i e_i^2$$

with displacement $\mathbf{u} = (u\ v)^\top$
and residual errors $e_i = f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i)$

Robust Error Metrics

- SSD metric is sensitive to outliers
- Solution: apply a (more) robust error metric

$$E_{\text{SRD}}(\mathbf{u}) = \sum_i \rho\left(f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i)\right) = \sum_i \rho(e_i)$$

Robust Error Metrics

- Sum of Absolute Differences

$$\rho_{\text{SAD}}(e) = |e|$$

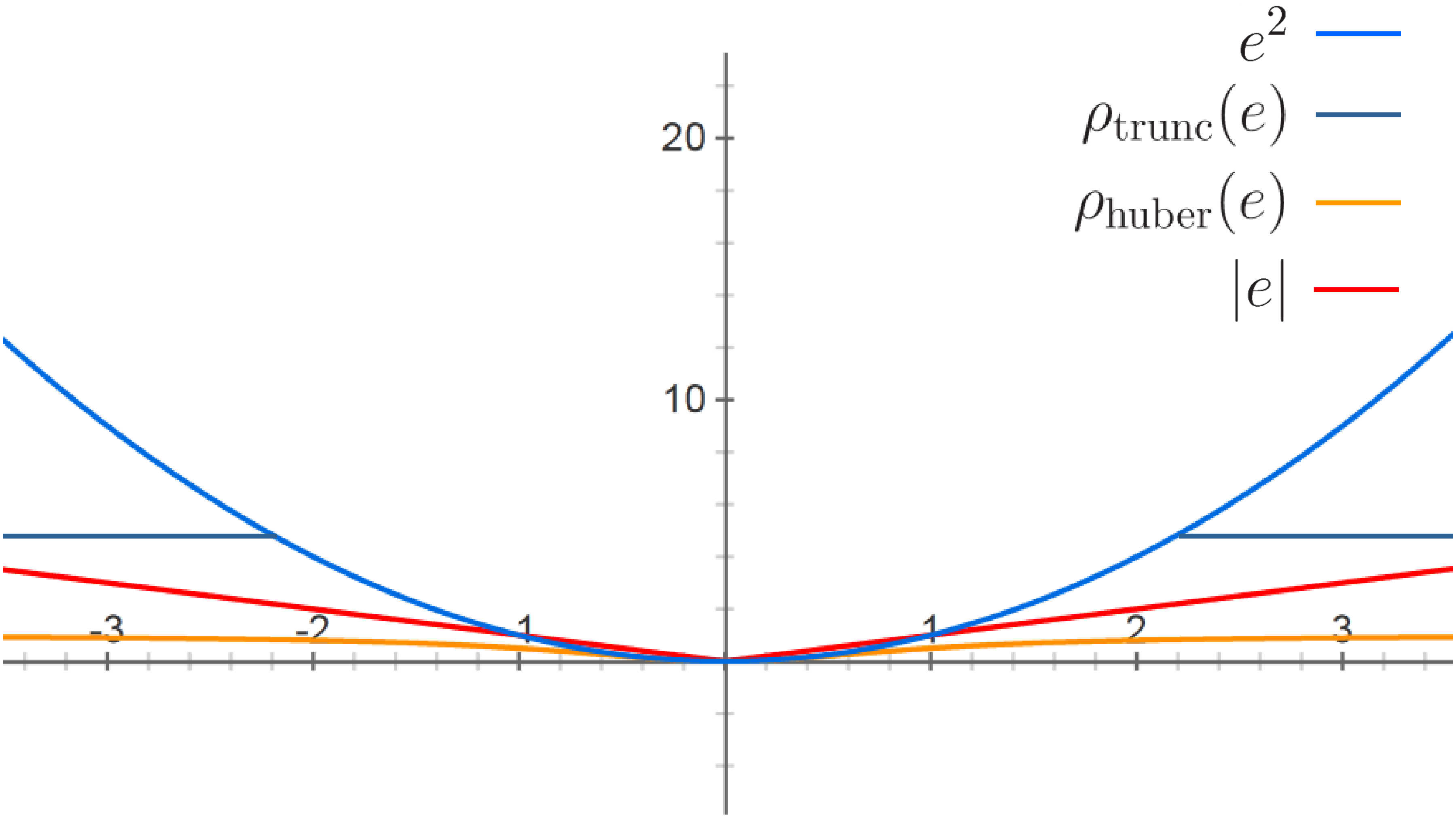
- Sum of truncated errors

$$\rho_{\text{trunc}}(e) = \begin{cases} e^2 & \text{if } |e| < b \\ b^2 & \text{otherwise} \end{cases}$$

- Geman-McClure function (Huber norm)

$$\rho_{\text{huber}}(e) = \frac{e^2}{1 + e^2/b^2}$$

Robust Error Metrics



Windowed SSD

- Images (and image patches) have finite size
- Standard SSD has a bias towards smaller overlaps (less error terms)
- Solution: divide by the overlap area
- Root mean square error

$$E_{\text{RMS}}(\mathbf{u}) = \sqrt{E_{\text{SSD}}/A}$$

Exposure Differences

- Images might be taken with different exposure (auto shutter, white balance, …)
- Bias and gain model

$$f_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha) f_0(\mathbf{x}) + \beta$$

- With SSD we get

$$E_{\mathrm{BG}}(\mathbf{u}) = \sum_i \left(f_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha) f_0(\mathbf{x}_i) + \beta\right)^2$$

$$= \sum_i \alpha f_0(\mathbf{x}) + \beta - e_i^2$$

Cross-Correlation

- Maximize the product (instead of minimizing the differences)

$$E_{\mathrm{CC}}(\mathbf{u}) = -\sum_i f_0(\mathbf{x}_i) f_1(\mathbf{x}_i + \mathbf{u})$$

- Normalized cross-correlation (between -1..1)

$$E_{\mathrm{NCC}}(\mathbf{u}) = -\sum_i \frac{(f_0(\mathbf{x}_i) - \mathrm{mean}\, f_0)(f_1(\mathbf{x}_i + \mathbf{u}) - \mathrm{mean}\, f_1)}{\sqrt{\mathrm{var}\, f_0 \mathrm{var}\, f_1}}$$

General Idea

1. Define an error metric $E(\mathbf{u})$ that defines how well the two images match given a motion vector
2. Find the motion vector with the lowest error

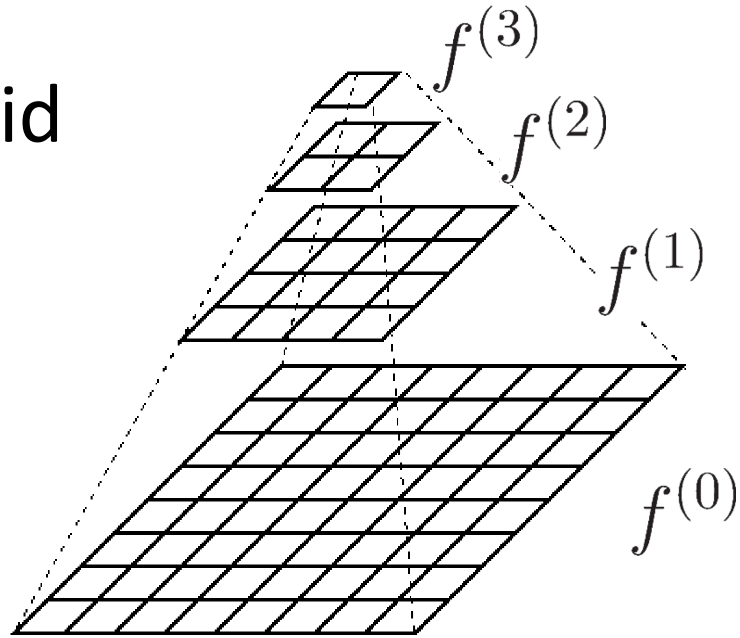
$$\mathbf{u}^* = \arg\min_{\mathbf{u}} E(\mathbf{u})$$

Finding the minimum

- Full search (e.g., ±16 pixels)
- Gradient descent
- Hierarchical motion estimation

Hierarchical motion estimation

- Construct image pyramid



$$f_k^{(l+1)}(\mathbf{x}_i) \leftarrow f_k^{(l)}(2\mathbf{x}_i)$$

- Estimate motion on coarse level
- Use as initialization for next finer level

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)}$$

Gradient Descent

- Perform gradient descent on the SSD energy function (Lucas and Kanade, 1981)
- Taylor expansion of energy function

$$E_{\mathrm{LK-SSD}}(\mathbf{u}+\Delta\mathbf{u}) = \sum_i (f_1(\mathbf{x}_i+\mathbf{u}+\Delta\mathbf{u}) - f_0(\mathbf{x}_i))^2$$

$$\approx \sum_i (f_1(\mathbf{x}_i+\mathbf{u}) + J_1(\mathbf{x}+\mathbf{u})\Delta\mathbf{u} - f_0(\mathbf{x}_i))^2$$

$$= \sum_i (J_1(\mathbf{x}+\mathbf{u})\Delta\mathbf{u} + e_i)^2$$

with $J_1(\mathbf{x}_i+\mathbf{u}) = \nabla f_1(\mathbf{x}_i+\mathbf{u}) = (\frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y})(\mathbf{x}_i+\mathbf{u})$

Least Squares Problem

- Goal: Minimize

$$E(\mathbf{u}+\Delta\mathbf{u}) = \sum_i (J_1(\mathbf{x}_i+\mathbf{u})\Delta\mathbf{u} + e_i)^2$$

- Solution: Compute derivative (and set to zero)

$$\frac{\partial E(\mathbf{u}+\Delta\mathbf{u})}{\partial\Delta\mathbf{u}} = 2A\Delta\mathbf{u} + 2\mathbf{b}$$

with $A = \sum_i J_1^\top(\mathbf{x}_i+\mathbf{u})J_1(\mathbf{x}+\mathbf{u})$

and $\mathbf{b} = \sum_i e_i J_1^\top(\mathbf{x}_i+\mathbf{u})$

Least Squares Problem

1. Compute A,b from image gradients using

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} \sum f_x f_t \\ \sum f_y f_t \end{pmatrix}$$

with $f_x = \frac{\partial f_1(\mathbf{x})}{\partial x}, f_y = \frac{\partial f_1(\mathbf{x})}{\partial y}$

and $f_t = \frac{\partial f_t(\mathbf{x})}{\partial t} [\approx f_1(\mathbf{x}) - f_0(\mathbf{x})]$

2. Solve $A\Delta\mathbf{u} = -\mathbf{b}$

$$\Rightarrow \quad \Delta\mathbf{u} = -A^{-1}\mathbf{b}$$

All of these computation are super fast!

Covariance of the Estimated Motion

- Assuming (small) Gaussian noise in the images

$$f_{\text{obs}}(\mathbf{x}_i) = f_{\text{true}}(\mathbf{x}_i) + \epsilon_i$$

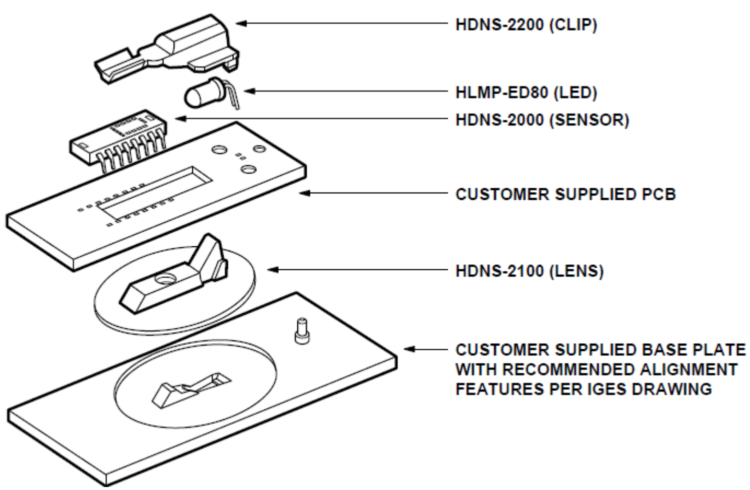
  with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

- … results in uncertainty in the motion estimate with covariance (e.g., useful for Kalman filter)

$$\Sigma_u = \sigma^2 A^{-1}$$

Optical Computer Mouse (since 1999)

- E.g., ADNS3080 from Agilent Technologies, 2005
  - 6400 fps
  - 30x30 pixels
  - 4 USD



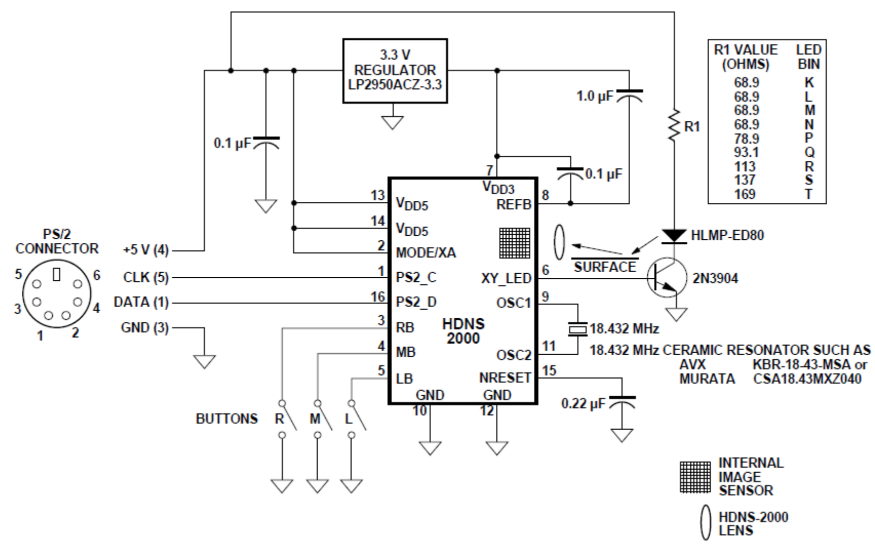
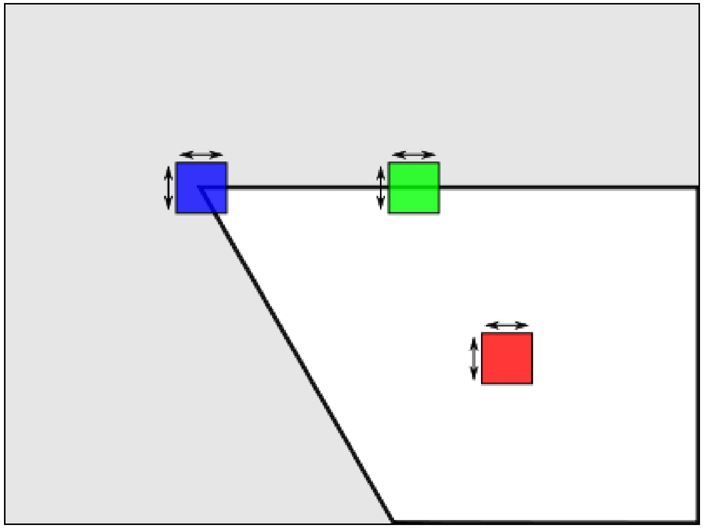


Image Patches

- Sometimes we are interested of the motion of a small image patches
- **Problem:** some patches are easier to track than others
- What patches are easy/difficult to track?
- How can we recognize "good" patches?

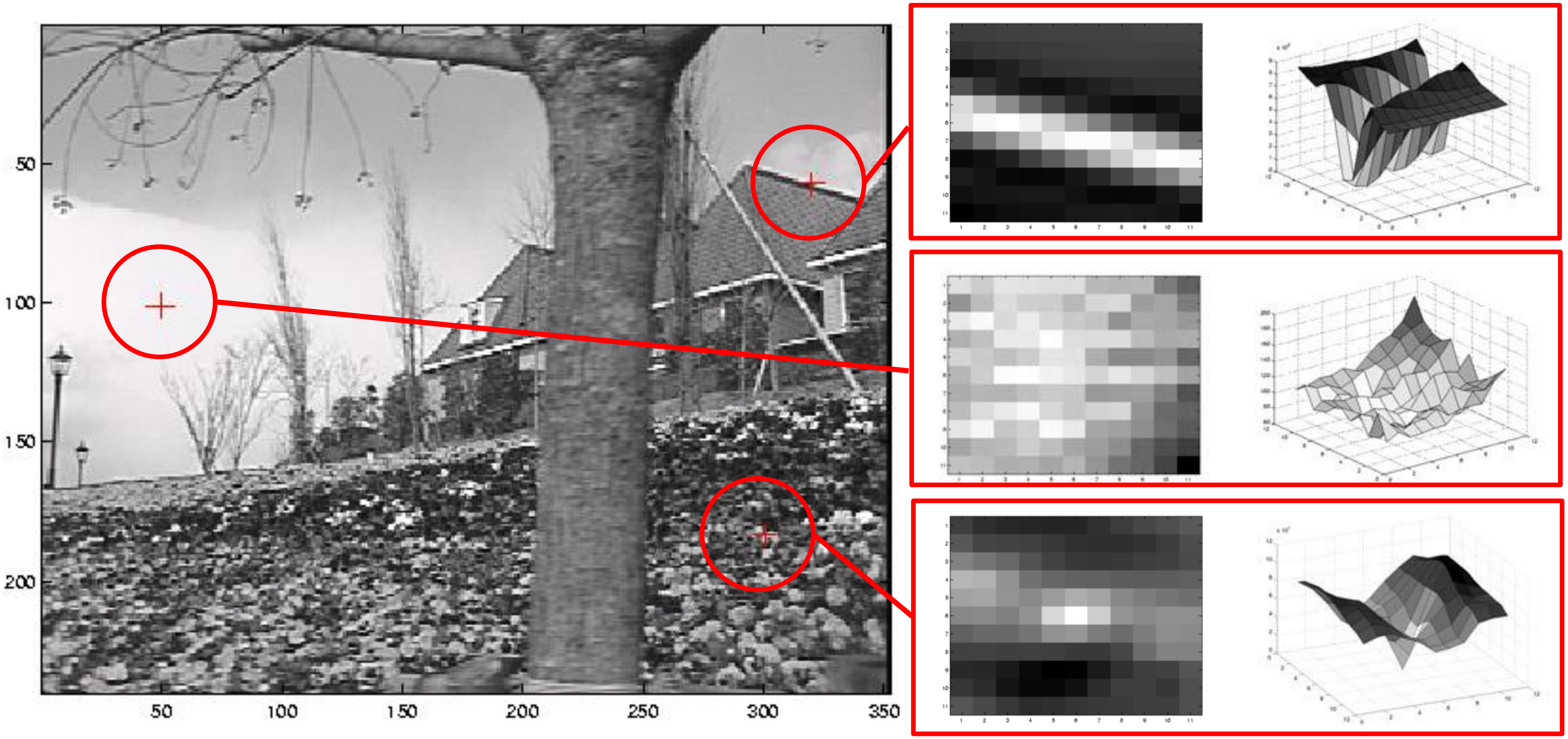
Image Patches

- Sometimes we are interested of the motion of a small image patches
- **Problem:** some patches are easier to track than others



Example

- Let's look at the shape of the energy functional



Corner Detection

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix}$$

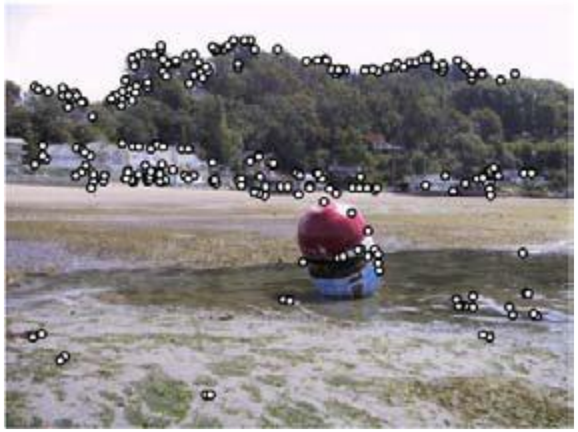
- **Idea:** Inspect eigenvalues λ_1, λ_2 of Hessian A
  - λ_1, λ_2 small → no point of interest
  - λ_1 large, λ_2 small → edge
  - λ_1, λ_2 large → corner
- Harris detector (does not need eigenvalues)

$$\lambda_1 \lambda_2 > \kappa (\lambda_1 + \lambda_2)^2 \Leftrightarrow \det(A) > \kappa \operatorname{trace}^2(A)$$

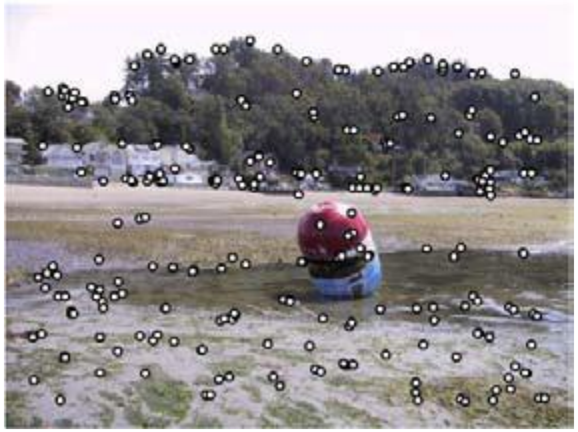
- Shi-Tomasi (or Kanade-Lucas) $\min(\lambda_1, \lambda_2) > \kappa$

Corner Detection

1. For all pixels, computer corner strength
2. Non-maximal suppression
   (E.g., sort by strength, strong corner suppresses weaker corners in circle of radius *r*)



strongest responses



non-maximal suppression

Other Detectors

- Förstner detector (localize corner with sub-pixel accuracy)
- FAST corners (learn decision tree, minimize number of tests → super fast)
- Difference of Gaussians / DoG (scale-invariant detector)
- …

Kanade-Lucas-Tomasi (KLT) Tracker

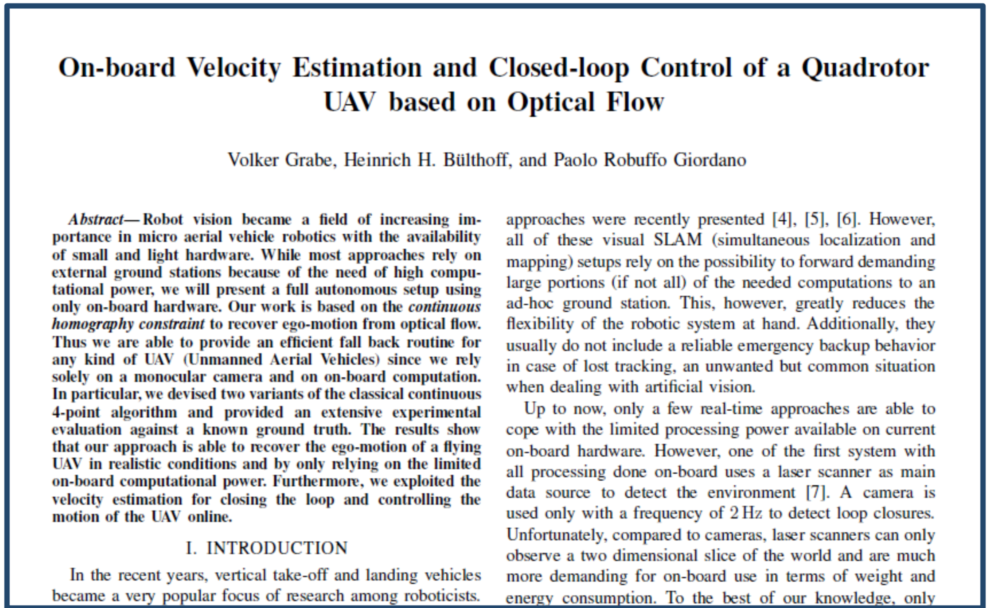
- Algorithm
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4
- KLT tracker is highly efficient (real-time on CPU) but provides only sparse motion vectors
- Dense optical flow methods require GPU

Example

3D Motion Estimation

(How) Can we recover the camera motion from the estimated flow field?

- Research paper: Grabe et al., ICRA 2012
  http://www9.in.tum.de/~sturmju/dirs/icra2012/data/papers/2025.pdf



On-board Velocity Estimation and Closed-loop Control of a Quadrotor UAV based on Optical Flow

Volker Grabe, Heinrich H. Bülthoff, and Paolo Robuffo Giordano

***Abstract*— Robot vision became a field of increasing importance in micro aerial vehicle robotics with the availability of small and light hardware. While most approaches rely on external ground stations because of the need of high computational power, we will present a full autonomous setup using only on-board hardware. Our work is based on the *continuous homography constraint* to recover ego-motion from optical flow. Thus we are able to provide an efficient fall back routine for any kind of UAV (Unmanned Aerial Vehicles) since we rely solely on a monocular camera and on on-board computation. In particular, we devised two variants of the classical continuous 4-point algorithm and provided an extensive experimental evaluation against a known ground truth. The results show that our approach is able to recover the ego-motion of a flying UAV in realistic conditions and by only relying on the limited on-board computational power. Furthermore, we exploited the velocity estimation for closing the loop and controlling the motion of the UAV online.**

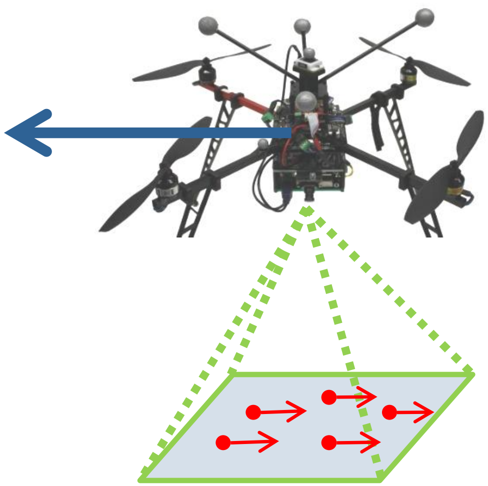
I. INTRODUCTION

In the recent years, vertical take-off and landing vehicles became a very popular focus of research among roboticists. approaches were recently presented [4], [5], [6]. However, all of these visual SLAM (simultaneous localization and mapping) setups rely on the possibility to forward demanding large portions (if not all) of the needed computations to an ad-hoc ground station. This, however, greatly reduces the flexibility of the robotic system at hand. Additionally, they usually do not include a reliable emergency backup behavior in case of lost tracking, an unwanted but common situation when dealing with artificial vision.

Up to now, only a few real-time approaches are able to cope with the limited processing power available on current on-board hardware. However, one of the first system with all processing done on-board uses a laser scanner as main data source to detect the environment [7]. A camera is used only with a frequency of 2 Hz to detect loop closures. Unfortunately, compared to cameras, laser scanners can only observe a two dimensional slice of the world and are much more demanding for on-board use in terms of weight and energy consumption. To the best of our knowledge, only

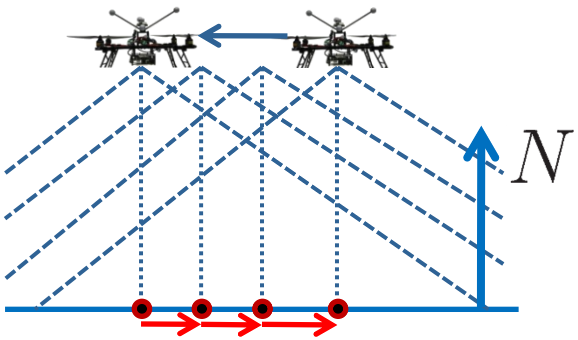
Approach [Grabe et al., ICRA'12]

- Compute optical flow
- Estimate homography between images
- Extract angular and (scaled) linear velocity
- Additionally employ information from IMU

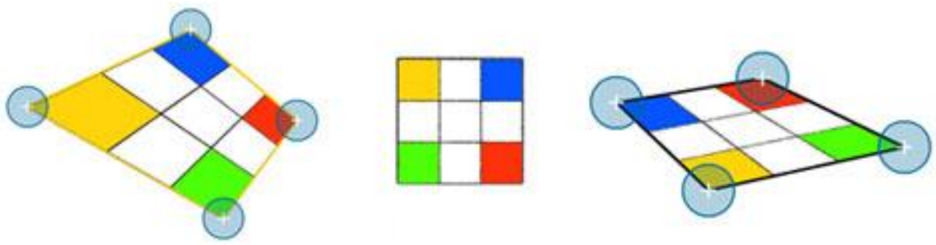


Assumptions

1. The quadrocopter moves slowly relative to the sampling rate
   → limited search radius



2. The environment is planar with normal N
   → image transformation is a homography



Apparent Velocity of a Point

- Stationary 3D point feature, given in camera frame

$$\mathbf{p} \in \mathbb{R}^3$$

- Moving camera with twist

$$\boldsymbol{\xi} = (\mathbf{v}^\top, \boldsymbol{\omega}^\top)^\top \in \mathbb{R}^6$$

- **Apparent velocity** of the point in camera frame

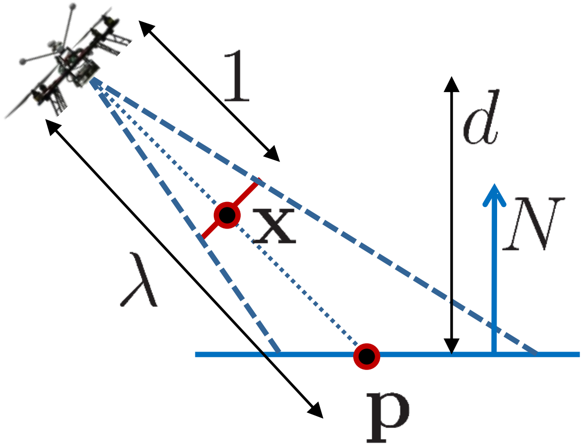
$$\dot{\mathbf{p}} = [\boldsymbol{\omega}]_\times \mathbf{p} + \boldsymbol{v}$$

Continuous Homography Matrix

- Assumption: All feature points are located on a plane

$$N^\top \mathbf{p} = d$$

with plane normal $N \in \mathbb{R}^3$
and distance $d \in \mathbb{R}$



Continuous Homography Matrix

- Rewrite this to $\frac{1}{d}N^\top \mathbf{p} = 1$ and plug it into the equation for the apparent velocity, we obtain

$$\dot{\mathbf{p}} = [\boldsymbol{\omega}]_\times \mathbf{p} + \boldsymbol{v}\frac{1}{d}N^\top \mathbf{p} = \underbrace{\left([\boldsymbol{\omega}]_\times + \boldsymbol{v}\frac{1}{d}N^\top\right)}_{H \in \mathbb{R}^{3\times 3}} \mathbf{p} = H\mathbf{p}$$

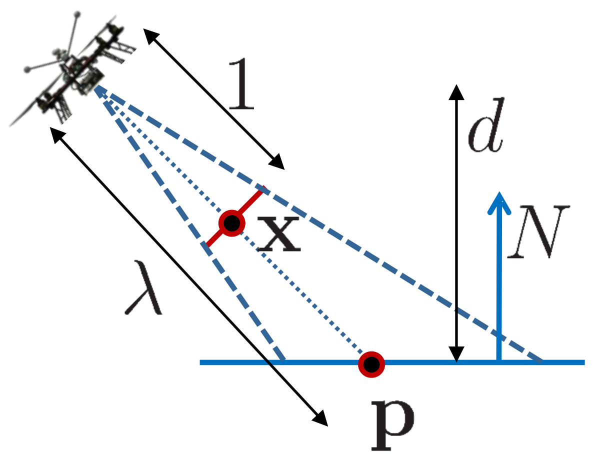
- H is called the **continuous homography matrix**
- Note: H contains both the linear/angular velocity $(\boldsymbol{v}, \boldsymbol{\omega})$ and the scene structure (N, d)

Continuous Homography Constraint

- The camera observes point $\mathbf{p} \in \mathbb{R}^3$ at pixel $\mathbf{x} \in \mathbb{R}^2$ (assuming $K = I$ for simplicity)

$$\tilde{\mathbf{x}} = \lambda \bar{\mathbf{x}} = \mathbf{p}$$

- The KLT tracker estimates the motion $\mathbf{u}$ of the feature track in the image
- Constraint: $\boxed{\mathbf{u} = \dot{\mathbf{x}}}$



Continuous Homography Constraint

- We now have
  1. $\dot{\mathbf{p}} = H\mathbf{p}$
  2. $\dot{\mathbf{p}} = \dot{\lambda}\bar{\mathbf{x}} + \lambda\bar{\mathbf{u}}$ (time derivative of $\mathbf{p} = \lambda\bar{\mathbf{x}}$ and the optical flow constraint $\mathbf{u} = \dot{\mathbf{x}}$)
- Let's combine these two formulas…

Continuous Homography Constraint

- Combining these formulas gives us

$$\dot{\lambda}\bar{\mathbf{x}} + \lambda\bar{\mathbf{u}} = H\mathbf{p}$$

$$\lambda\bar{\mathbf{u}} = H\mathbf{p} - \dot{\lambda}\bar{\mathbf{x}}$$

$$\bar{\mathbf{u}} = H\bar{\mathbf{x}} - \frac{\dot{\lambda}}{\lambda}\bar{\mathbf{x}}$$

- Multiply both sides with $[\bar{\mathbf{x}}]_\times$ gives us

$$[\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}} - \underbrace{[\bar{\mathbf{x}}]_\times\frac{\dot{\lambda}}{\lambda}\bar{\mathbf{x}}}_{=0}$$

$$\Rightarrow [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}}$$

Approach

- **Result:** For all observed motions in the image, the continuous homography constraint holds

$$[\bar{\mathbf{x}}]_\times \bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H \bar{\mathbf{x}}$$

- How can we use this to estimate the camera motion?!

Approach

- **Result:** For all observed motions in the image, the continuous homography constraint holds

$$[\bar{\mathbf{x}}]_\times \bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H \bar{\mathbf{x}}$$

- How can we use this to estimate the camera motion?
  1. Estimate H from at least 4 feature tracks
  2. Recover $(\boldsymbol{v}, \boldsymbol{\omega})$ and (N, d) from H

Remember: $H = [\boldsymbol{\omega}]_\times + \boldsymbol{v}\frac{1}{d}N^\top$

Step 1: Estimate H

- Continuous homography constraint

$$[\bar{\mathbf{x}}]_\times H \bar{\mathbf{x}} = [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}$$

- Stack matrix H as a vector $\mathbf{h} \in \mathbb{R}^9$ and rewrite

$$M^\top \mathbf{h} = [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}$$

→Linear system of equations

- For several feature tracks

$$\begin{pmatrix} M_1^\top \\ M_2^\top \\ \vdots \end{pmatrix} \mathbf{h} = \begin{pmatrix} [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}_1^\top \\ [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}_2^\top \\ \vdots \end{pmatrix}$$

Step 1: Estimate H

- Linear set of equations

$$\underbrace{\begin{pmatrix} M_1^\top \\ M_2^\top \\ \vdots \end{pmatrix}}_{A} \mathbf{h} = \underbrace{\begin{pmatrix} [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}_1^\top \\ [\bar{\mathbf{x}}]_\times \bar{\mathbf{u}}_2^\top \\ \vdots \end{pmatrix}}_{\mathbf{b}}$$

- Solve for $\mathbf{h}$ using least squares

$$A\mathbf{h} = \mathbf{b}$$

$$\Rightarrow \mathbf{h} = (A^\top A)^{-1} A^\top \mathbf{b}$$

Step 2: Recover camera motion

Grabe et al. investigated three alternatives:

1. Recover $(\boldsymbol{\omega}, \frac{\boldsymbol{v}}{d}, N)$ from $H = [\boldsymbol{\omega}]_\times + \boldsymbol{v}\frac{1}{d}N^\top$ using the 8-point algorithm (not yet explained)
2. Use angular velocity $\boldsymbol{\omega}$ from IMU to de-rotate observed feature tracks beforehand, then:
$$H = \boldsymbol{v}\frac{1}{d}N^\top$$
3. Additionally use gravity vector from IMU as plane normal $N = N_{\mathrm{IMU}}$, then
$$\frac{\boldsymbol{v}}{d} = H(N^\top N)^{-1}$$

Evaluation

- Comparison of estimated velocities with ground truth from motion capture system

Algorithm	Norm error	Std. deviation
Pure vision	0.134 $\frac{m}{s}$	0.094 $\frac{m}{s}$
Ang. vel. known	0.117 $\frac{m}{s}$	0.093 $\frac{m}{s}$
Normal known	0.113 $\frac{m}{s}$	0.088 $\frac{m}{s}$

- Comparison of actual velocity with desired velocity (closed-loop control)

Algorithm	Norm error	Std. deviation
Pure vision	0.084 $\frac{m}{s}$	0.139 $\frac{m}{s}$
Ang. vel. known	0.039 $\frac{m}{s}$	0.042 $\frac{m}{s}$
Normal known	0.028 $\frac{m}{s}$	0.031 $\frac{m}{s}$

Visual Velocity Control

- All computations are carried out on-board (18fps)

[Grabe et al., ICRA '12]

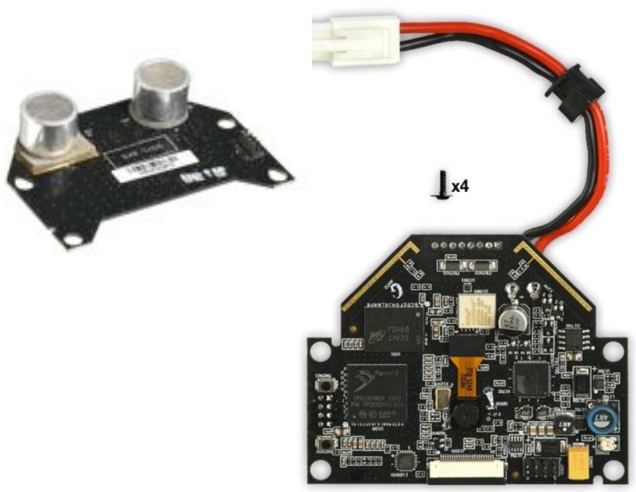
Landing on a Moving Platform

- Similar approach, but with offboard computation

[Herissé et al., T-RO ’12]

Commercial Solutions

- Helicommand 3D from Robbe
  2(?) cameras, IMU, air pressure sensor, 450 EUR
- Parrot Mainboard + Navigation board
  1 camera, IMU, ultrasound sensor, 210 USD



Lessons Learned Today

- How to estimate the translational motion from camera images
- Which image patches are easier to track than others
- How to estimate 3D motion from multiple feature tracks (and IMU data)

A Few Ideas for Your Mini-Project

- Person following (colored shirt or wearing a marker)
- Flying camera for taking group pictures (possibly using the OpenCV face detector)
- Fly through a hula hoop (brightly colored, white background)
- Navigate through a door (brightly colored)
- Navigate from one room to another (using ground markers)
- Avoid obstacles using optical flow
- Landing on a moving platform
- **Your own idea here – be creative!**
- ...

Joggobot

- Follows a person wearing a visual marker

[http://exertiongameslab.org/projects/joggobot]
`