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

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

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

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

$$
\mathbf{p}(t) = R(t)\mathbf{p}(0) + \mathbf{t}(t)
= \exp([\mathbf{\omega}]_\times t)\mathbf{p}(0) + \mathbf{v}t
$$
Angular and linear velocities

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

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

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

$\dot{\mathbf{p}}(0) = [\omega]_\times \mathbf{p}(0) + \mathbf{v}$
Recap: Perspective Projection
3D to 2D Perspective Projection

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

\[ \tilde{x} = \lambda \tilde{x} = p \]

- Remember, \( \tilde{x} \) is homogeneous, need to normalize

\[ \tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \quad \Rightarrow \quad x = \begin{pmatrix} \tilde{x} / \tilde{z} \\ \tilde{y} / \tilde{z} \end{pmatrix} \]
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

\[
\tilde{x} = \begin{pmatrix}
    f_x & s & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0
\end{pmatrix}
\tilde{p}
\]

- Focal length \( f_x, f_y \)
- Camera center \( c_x, c_y \)
- Skew \( s \)
Image Plane

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

Example:
- Discrete case $x \in [0, W) \times [0, H) \subset \mathbb{N}_0^2$ (default in this course)
- Continuous case $x \in [0, 1] \times [0, 1] \subset \mathbb{R}^2$
We can think of an image as a function $f : \Omega \rightarrow \mathbb{R}$

$f(x)$ gives the intensity at position $x$

Color images are vector-valued functions

$$f(x) = \begin{pmatrix} r(x) \\ g(x) \\ b(x) \end{pmatrix}$$
Realistically, the image function is only defined on a rectangle and has finite range

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

Image can be represented as a matrix

Alternative notations

\[ F_{ij}, f(i, j), f(x, y), f(x), \ldots \]

often (row,column)

often (column,row)
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)
  \[ V = B^{\frac{1}{\gamma}} \quad \text{with} \quad \gamma = 2.2 \]
- CRTs / monitors do the inverse \[ B = V^\gamma \]
- Almost all images are gamma compressed
  \[ \Rightarrow \text{Double brightness results only in a 37\% higher intensity value (!)} \]
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

\[ G = HF \]
Spatially Invariant Filtering

- We are often interested in spatially invariant operations

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

- Example

\[
\begin{array}{ccc}
111 & 115 & 113 \\
135 & 138 & 137 \\
163 & 168 & 188 \\
180 & 184 & 206 \\
189 & 193 & 214 \\
191 & 201 & 217 \\
195 & 205 & 216 \\
199 & 203 & 223 \\
\end{array}
\]

\[
\begin{array}{ccc}
-1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
\end{array}
\]

\[
\begin{array}{c}
? \\
\end{array}
\]
Spatially Invariant Filtering

- We are often interested in spatially invariant operations

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

- Example

\[
\begin{array}{ccc}
111 & 115 & 113 \\
135 & 138 & 137 \\
163 & 168 & 188 \\
180 & 184 & 206 \\
189 & 193 & 216 \\
191 & 201 & 220 \\
195 & 205 & 222 \\
199 & 203 & 228 \\
\end{array}
\begin{array}{ccc}
111 & 112 & 111 \\
111 & 145 & 146 \\
149 & 146 \\
196 & 202 & 206 \\
202 & 200 \\
195 & 193 \\
113 & 68 \\
108 & 68 \\
\end{array}
\begin{array}{ccc}
* & 1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
\end{array}
\begin{array}{ccc}
? & ? & ? \\
? & -5 & 9 \\
? & -29 & 18 \\
? & -50 & 40 \\
? & -41 & 41 \\
? & -24 & 37 \\
? & -23 & 33 \\
\end{array}
\begin{array}{ccc}
? & ? & ? \\
? & -9 & 21 \\
? & 24 & 4 \\
? & 142 & 349 \\
? & -175 & -217 \\
? & -120 & -224 \\
? & -134 & -360 \\
\end{array}
\begin{array}{ccc}
? & ? & ? \\
-12 & -7 \\
-34 & -71 \\
-88 & -175 \\
-224 & -120 \\
-217 & -360 \\
? & ? \\
\end{array}
\begin{array}{ccc}
? & ? & ? \\
10 & 5 \\
10 & 4 \\
10 & 0 \\
10 & -10 \\
10 & -23 \\
? & ? \\
\end{array}
\]
Important Filters

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

Dr. Jürgen Sturm, Computer Vision Group, TUM
\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
Image shift (translation)

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
Image rotation

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
Image rotation

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

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

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
Rectangular Filter

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
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

Visual Navigation for Flying Robots
The image gradient $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^\top$ points in the direction of increasing intensity (steepest ascend)
The image gradient $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^\top$ 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)

\[ \theta = \text{atan2} \left( \frac{\partial f}{\partial y}, \frac{\partial f}{\partial x} \right) \]

- Gradient magnitude (edge strength)

\[ \| \nabla f \| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2} \]
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: \([-0.5, 0, 0.5]\)
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**: \([-1 \mid 1]\)
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

\[ g(i, j) = f \ast h = \sum_{k,l} h(i - k, j - l) f(k, l) \]
Example

\[ g(i, j) = f \ast h = \sum_{k,l} h(i-k, j-l) f(k, l) \]
(Dense) Motion Estimation

- 2D motion
- 3D motion
Problem Statement

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

For the moment, let’s assume that the camera only moves in the xy-plane, i.e., $u = (u \ v)^\top$

- Extension to 3D follows
General Idea

1. Define an error metric $E(u)$ that defines how well the two images match given a motion vector.

2. Find the motion vector with the lowest error

$$u^* = \arg \min_u E(u)$$
Error Metrics for Image Comparison

- Sum of Squared Differences (SSD)

\[ E_{SSD}(u) = \sum_i \left( f_1(x_i + u) - f_0(x_i) \right)^2 = \sum_i e_i^2 \]

with displacement \( u = (u \ v)^	op \)

and residual errors \( e_i = f_1(x_i + u) - f_0(x_i) \)
Robust Error Metrics

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

\[ E_{SRD}(u) = \sum_i \rho \left( f_1(x_i + u) - f_0(x_i) \right) = \sum_i \rho(e_i) \]
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

- $e^2$
- $\rho_{\text{trunc}}(e)$
- $\rho_{\text{huber}}(e)$
- $|e|$
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

\[ E_{\text{RMS}}(u) = \sqrt{\frac{E_{\text{SSD}}}{A}} \]
Exposure Differences

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

\[ f_1(x + u) = (1 + \alpha)f_0(x) + \beta \]

- With SSD we get

\[ E_{BG}(u) = \sum_i (f_1(x_i + u) - (1 + \alpha)f_0(x_i) + \beta)^2 \]

\[ = \sum_i \alpha f_0(x) + \beta - e_i^2 \]
Cross-Correlation

- Maximize the product (instead of minimizing the differences)

\[ E_{CC}(u) = - \sum_{i} f_0(x_i) f_1(x_i + u) \]

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

\[ E_{NCC}(u) = \frac{- \sum_{i} (f_0(x_i) - \text{mean} f_0)(f_1(x_i + u) - \text{mean} f_1)}{\sqrt{\text{var} f_0 \text{var} f_1}} \]
**General Idea**

1. Define an error metric $E(u)$ that defines how well the two images match given a motion vector.

2. Find the motion vector with the lowest error

$$u^* = \arg \min_u E(u)$$
Finding the minimum

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

- Construct image pyramid

$ f_k^{(l+1)}(x_i) \leftarrow f_k^{(l)}(2x_i) $

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

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

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

\[ E_{\text{LK-SSD}}(u + \Delta u) = \sum_i (f_1(x_i + u + \Delta u) - f_0(x_i))^2 \]

\[ \approx \sum_i (f_1(x_i + u) + J_1(x + u)\Delta u - f_0(x_i))^2 \]

\[ = \sum_i (J_1(x + u)\Delta u + e_i)^2 \]

with \( J_1(x_i + u) = \nabla f_1(x_i + u) = \left( \frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y} \right)(x_i + u) \)
Least Squares Problem

- **Goal**: Minimize

\[
E(u + \Delta u) = \sum_i (J_1(x_i + u)\Delta u + e_i)^2
\]

- **Solution**: Compute derivative (and set to zero)

\[
\frac{\partial E(u + \Delta u)}{\partial \Delta u} = 2A\Delta u + 2b
\]

with

\[
A = \sum_i J_1^\top(x_i + u)J_1(x + u)
\]

and

\[
b = \sum_i e_i J_1^\top(x_i + u)
\]
Least Squares Problem

1. Compute $A, b$ from image gradients using

\[
A = \begin{pmatrix}
\sum f_x^2 & \sum f_x f_y \\
\sum f_x f_y & \sum f_y^2
\end{pmatrix} \\
b = \begin{pmatrix}
\sum f_x f_t \\
\sum f_y f_t
\end{pmatrix}
\]

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

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

2. Solve $A \Delta u = -b$

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

All of these computations are super fast!
Covariance of the Estimated Motion

- Assuming (small) Gaussian noise in the images

\[ f_{\text{obs}}(x_i) = f_{\text{true}}(x_i) + \epsilon_i \]

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 in 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 in the motion of 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 \( \lambda_1, \lambda_2 \) of Hessian \( A \)
  - \( \lambda_1, \lambda_2 \) small \( \rightarrow \) no point of interest
  - \( \lambda_1 \) large, \( \lambda_2 \) small \( \rightarrow \) edge
  - \( \lambda_1, \lambda_2 \) large \( \rightarrow \) corner

- Harris detector (does not need eigenvalues)
  \[ \lambda_1 \lambda_2 > \kappa (\lambda_1 + \lambda_2)^2 \iff \det(A) > \kappa \text{ trace}^2(A) \]

- Shi-Tomasi (or Kanade-Lucas)
  \[ \min(\lambda_1, \lambda_2) > \kappa \]
Corner Detection

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

![Strongest responses](image1.png) ![Non-maximal suppression](image2.png)
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
(How) Can we recover the camera motion from the estimated flow field?

- Research paper: Grabe et al., ICRA 2012
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
  \[ \xi = (\mathbf{v}^\top, \mathbf{\omega}^\top)^\top \in \mathbb{R}^6 \]

- Apparent velocity of the point in camera frame
  \[ \dot{\mathbf{p}} = [\mathbf{\omega}] \times \mathbf{p} + \mathbf{v} \]
Continuous Homography Matrix

- Assumption: All feature points are located on a plane

\[ N^T p = d \]

with plane normal \( N \in \mathbb{R}^3 \) and distance \( d \in \mathbb{R} \)
Continuous Homography Matrix

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

$$\dot{p} = [\omega]_\times p + \frac{1}{d}N^\top p = \left( [\omega]_\times + \frac{1}{d}N^\top \right) p = H p$$

- $H$ is called the continuous homography matrix
- Note: $H$ contains both the linear/angular velocity $(\nu, \omega)$ and the scene structure $(N, d)$
Continuous Homography Constraint

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

\[
\tilde{x} = \lambda \bar{x} = p
\]

- The KLT tracker estimates the motion \( u \) of the feature track in the image

- Constraint: \( u = \dot{x} \)
Continuous Homography Constraint

- We now have
  1. \( \dot{p} = Hp \)
  2. \( \dot{p} = \lambda \dot{x} + \lambda \dot{u} \) (time derivative of \( p = \lambda x \) and the optical flow constraint \( u = \dot{x} \))

- Let’s combine these two formulas...
Continuous Homography Constraint

- Combining these formulas gives us
  \[
  \lambda \ddot{x} + \lambda \ddot{u} = H p
  \]
  \[
  \lambda \ddot{u} = H p - \lambda \dot{x}
  \]
  \[
  \ddot{u} = H \dot{x} - \frac{\lambda}{\lambda} \dot{x}
  \]

- Multiply both sides with \([\bar{x}] \times\) gives us
  \[
  [\bar{x}] \times \ddot{u} = [\bar{x}] \times H \dot{x} - [\bar{x}] \times \frac{\lambda}{\lambda} \dot{x}
  \]
  \[
  \Rightarrow [\bar{x}] \times \ddot{u} = [\bar{x}] \times H \dot{x}
  \]
Approach

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

\[
[\tilde{x}] \times \tilde{u} = [\tilde{x}] \times H\tilde{x}
\]

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

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

\[
[\bar{x}] \times \bar{u} = [\bar{x}] \times H \bar{x}
\]

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

1. Estimate \( H \) from at least 4 feature tracks
2. Recover \((v, \omega)\) and \((N, d)\) from \( H \)

Remember:

\[
H = [\omega] \times + v_{d}N^{\top}
\]
Step 1: Estimate H

- Continuous homography constraint
  \[
  [\bar{x}] \times H \bar{x} = [\bar{x}] \times \bar{u}
  \]

- Stack matrix H as a vector \( h \in \mathbb{R}^9 \) and rewrite
  \[
  M^T h = [\bar{x}] \times \bar{u}
  \]

→ Linear system of equations

- For several feature tracks
  \[
  \begin{pmatrix}
  M_1^T \\
  M_2^T \\
  \vdots
  \end{pmatrix}
  h = \begin{pmatrix}
  [\bar{x}] \times \bar{u}_1^T \\
  [\bar{x}] \times \bar{u}_2^T \\
  \vdots
  \end{pmatrix}
  \]
Step 1: Estimate $H$

- Linear set of equations

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

- Solve for $h$ using least squares

$$Ah = b$$

$$\Rightarrow h = (A^\top A)^{-1} A^\top b$$
Step 2: Recover camera motion

Grabe et al. investigated three alternatives:

1. Recover \((\omega, \frac{v}{d}, N)\) from \(H = [\omega]_x + \frac{v}{d} N^\top\) using the 8-point algorithm (not yet explained)

2. Use angular velocity \(\omega\) from IMU to de-rotate observed feature tracks beforehand, then:
   \[H = \frac{v}{d} N^\top\]

3. Additionally use gravity vector from IMU as plane normal \(N = N_{IMU}\), then
   \[\frac{v}{d} = H(N^\top N)^{-1}\]
Evaluation

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Norm error</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure vision</td>
<td>0.134 m/s</td>
<td>0.094 m/s</td>
</tr>
<tr>
<td>Ang. vel. known</td>
<td>0.117 m/s</td>
<td>0.093 m/s</td>
</tr>
<tr>
<td>Normal known</td>
<td>0.113 m/s</td>
<td>0.088 m/s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Norm error</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure vision</td>
<td>0.084 m/s</td>
<td>0.139 m/s</td>
</tr>
<tr>
<td>Ang. vel. known</td>
<td>0.039 m/s</td>
<td>0.042 m/s</td>
</tr>
<tr>
<td>Normal known</td>
<td>0.028 m/s</td>
<td>0.031 m/s</td>
</tr>
</tbody>
</table>
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

- Helicommmand 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]