Hough Transform

$y = ax + b$

$y = \left(-\frac{\cos\theta}{\sin\theta}\right) \cdot x + \frac{r}{\sin(\theta)}$

$\Rightarrow r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$

$\Rightarrow$ family of lines that go through (x_0, y_0) is $r = x_0 \cdot \cos\theta + y_0 \sin\theta$

$\Downarrow$

Each pair (r, θ) represents one line in that family.

Plotting every pair (r, θ) gives a sinusoid. A line that goes through p_0 and p_1 is represented by the point where two sinusoids cross. If there are many points in a line, there'll be many sinusoids crossing in some $p_{\sigma r}$. (If the line is not going exactly through the points, there are many nearby intersections.). $p_{\sigma r}$ can be read as a line.

Circles can work very similarly: $r^2 = (x-a)^2 + (y-b)^2$. A circle through a point p_1 is determined by its radius and center x & y. i.e. by r, a, b for fixed x, y. $\Rightarrow$ Family of circles $\Rightarrow$ where two families intersect, there's a circle that contains both points. (r, a, b) gives that circle.

Rigid Bodies

Center of Mass $= \frac{1}{M} \sum_i m_i r_i$ ← total mass

$p(t) = Rot(t)\, p_0 + x(t) = T \cdot R \cdot p_0$

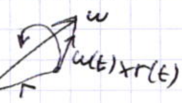
Linear Velocity: $v(t) = \frac{d}{dt} x(t) = x'(t)$

Angular Velocity: magnitude of $\omega(t)$, rotate around direction of $\omega(t)$

Rotation Matrix: $R' = [x' \; y' \; z']$ ← time derivatives of world space axis

// R' because R is the still rotation (orientation) //

If $\vec{r}(t)$ is bound to rotation axis $\omega(t)$, then $r' = \omega \times r$



crossproduct

$R' = \begin{bmatrix} \omega(t) \times \begin{pmatrix} R_{xx} \\ R_{xy} \\ R_{xz} \end{pmatrix}, & \dots \end{bmatrix}$

Or as quaternion $q'(t) = \frac{1}{2} \omega(t) q(t)$

Force $F = \sum_i F_i = \sum_i m_i a_i = ma$ // at Center of Mass

Torque $\tau = (r(t) - x(t)) \times F$ // not from COM but now also applied there

Linear Momentum: $p(t) = m \cdot v(t)$, Force $F = p'(t)$

Angular Momentum: $L(t) = I(t)\, \omega(t)$, Torque $\tau(t) = L'(t)$

← Moment of inertia, 3×3 matrix (or tensor)

$$I(t) = \sum_i \begin{pmatrix} m_i(r_{iy}'^2 + r_{iz}'^2) & -m_i(r_{ix}' r_{iy}') & -m_i(r_{ix}' r_{iz}') \\ \text{symmetrisch} & m_i(r_{ix}'^2 + r_{iz}'^2) & -m_i(r_{iy}' r_{iz}') \\ & & m_i(r_{ix}'^2 + r_{iy}'^2) \end{pmatrix}$$

can be precomputed in body space I_{body}

$I(t) = R(t) I_{body} R(t)^T$

Some Question

$\frac{d\,\text{center of Mass}}{dt}$: $x'(t) = \frac{P(t)}{m}$

$p'(t) = F_i(t)$

$q'(t) = \frac{1}{2}(I^{-1}(t) L(t))\, q(t)$

$L'(t) = (r_i(t) - x(t)) \times F_i(t)$

Question was to express the ODEs that govern the dynamics on a rigid body with a single force $F_i(t)$ applied to the point $r_i(t)$, assuming known mass and moment of inertia I

Collision handling

Impulse driven
- \+ fast simulation
- \+ does not require change in timestep
- \+ Instantaneous reaction
  Instant change in velocity stops collision
- \- Difficult to compute (correct change in Momentum)
- \- Complex for multiple collisions simultaneously (increasing)
  If multiple collide, have to solve a combined system

Force driven
- \+ Easy to compute: Just add forces
- \+ Collision responses are independent
- \- Slow simulation, may require smaller timestep for stability
- \- Slow response. Only no collision after some time
  At least 1 timestep needed to propagate from forces to velocities.

Homogenous Translation

$$\begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix}$$

Homog. Scaling

$$\begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix}$$

Homog. Rot. around Origin

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos & -\sin & 0 \\ 0 & \sin & \cos & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =: Rot_x$$

Rotation in y-z

$$\begin{Bmatrix} 1 & 0 & 0 \\ 0 & \cos & -\sin \\ 0 & \sin & \cos \end{Bmatrix}$$ if this is reversed, the angle is negated

Homog. Rot. around Point $\binom{a}{b}$

$$Trans\binom{a}{b} \cdot Rot_0(\varphi) \cdot Trans\binom{-a}{-b} = \begin{bmatrix} \cos & -\sin & -a\cdot\cos + b\cdot\sin + a \\ \sin & \cos & -a\cdot\sin - b\cdot\cos + b \\ 0 & 0 & 1 \end{bmatrix}$$

$Trans^{-1}_{a,b} = Trans\binom{-a}{-b}$

$Rot(\theta)^{-1} = Rot(-\theta)$

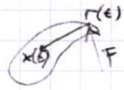
$Scale(a,b)^{-1} = Scale(\frac{1}{a}, \frac{1}{b})$

$$Trans\begin{pmatrix}a\\b\\c\end{pmatrix} \cdot Rot_x = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & \cos & -\sin & b \\ 0 & \sin & \cos & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans\begin{pmatrix}a\\b\\c\end{pmatrix} \cdot Rot_x(\varphi) \cdot Trans\begin{pmatrix}a\\b\\c\end{pmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & -\cos(\varphi)\cdot b + b + c\cdot\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) & -\cos(\varphi)\cdot c + c - b\cdot\sin(\varphi) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If vector $\vec{r}(t)$ is bound to a rotation axis, then $\vec{r}'(t) = \omega \times \vec{r}$ (ω: Angular Velocity)

$\Rightarrow R = \left((\omega \times column_1), (\omega \times col_2), (\omega \times col_3)\right)$



$F = ma = mv' = mx''$

$torque = \overrightarrow{(r-x)} \times \vec{F}$

Linear Momentum $p = m\,v(t) \to F = p'(t)$

Angular " $L(t) = I(t)\omega(t) \to \tau(t) = L'(t)$

($I(t)$: Moment of Inertia)

Compute $I(t)$ once in the object space and then rotate into world space

$I(t) = R(t) I_{body} R(t)^T$

// $I(t)$ is a big (3×3) matrix of masses
// and distances to COMass

RGB → XYZ

	R	G	B
X	0.64	0.3	0.15
Y	0.33	0.6	0.06
Z	0.03	0.1	0.79

$Z = 1 - X - Y$

Transform with calibration: $\begin{bmatrix} 0.64C_R & 0.3C_G & 0.15C_B \\ 0.33C_R & 0.6C_G & 0.06C_B \\ 0.03C_R & 0.1C_G & 0.79C_B \end{bmatrix} \Big(\Big)_{RGB} = \Big(\Big)_{XYZ}$

sRGB → YUV (PaL)

	R	G	B
Y	0.21	0.71	0.72
U	-0.1	-0.55	0.65
V	0.69	-0.62	-0.06

(Zahlen hier abgeschnitten)

CIE XYZ ⇒ xyY

$x = \frac{X}{X+Y+Z}$, $y = \frac{Y}{X+Y+Z}$

CIE xyY ⇒ XYZ

$X = \frac{Y}{y} x$

$Z = \frac{Y}{y}(1 - x - y)$

$\left(y = \frac{Y}{X+Y+Z}\right)$ similar for x and z

"Chromacity"

// big Y is the same in CIE_{xyY} and in XYZ

// Y stands for green AND Luminance

Wavelength form CIE Chart: Connect color and whitepoint, read where line cuts border ⇒ Dominant wavelength (unless it's opposite side)

Saturation is isoline with constant distance to border

Quaternions

$q'(t) = \frac{1}{2}\omega(t) q(t)$ when q is a rotation expressed as a unit quaternion

$i^2 = j^2 = k^2 = ijk = -1$, $ij = k$, $ji = -k$, $ik = -j$, $ki = j$, $jk = i$, $kj = -i$

$z_1 z_2 = s_1 s_2 - v_1 \cdot v_2 + s_1 v_2 + s_2 v_1 + \underbrace{v_1 \times v_2}_{\text{not same as } z_2 z_1}$

$z = s + v$, $\bar{z} = s - v$

$z\bar{z} = \|z\|^2$, $z^{-1} = \frac{\bar{z}}{\|z\|^2}$, $1 = zz^{-1} = z^{-1}z$

Translation is addition

Unit/Rotation-Quaternion: $q = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2}) \cdot u$

u: unit vector $= (0 + 0i + j + 0k)$ for example = rotation axis

$p' = qp\bar{q}^{-1} = Rp$

$$R = \begin{bmatrix} 1-2s(b^2+c^2) & 2s(ab-rc) & 2s(ac+br) \\ 2s(ab+cr) & 1-2s(a^2+c^2) & 2s(bc-ar) \\ 2s(ac-br) & 2s(bc+ar) & 1-2s(a^2+b^2) \end{bmatrix}$$

for $p = r + ai + bj + ck$, $s \equiv \|q\|^{-2}$ (=1 for unit). R is unitary

"Does q describe a rotation?" A: just bring it into form $q = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2}) \cdot \vec{u}$ (unit vector) if that works, it is

Eric Mink

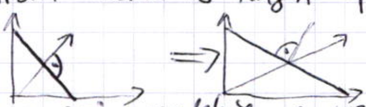
Shaders

Normals $n' = (M^{-1})^T n$ when point moved by $p' = Mp$ (reason: $\overbrace{(A\cdot n)}^{n'} \cdot \overbrace{(B \cdot p)}^{p'} \overset{=0}{=} 0$)

rgb [0,1] must be scaled to [-1,1]

Translation is ignored when transforming from world to tangent space.

Example where normal misbehaves:



if vertex shader contains gl_Position = proj * modelview * vec4(inPosition, 1.0f), then proj transforms points from world space to screen space and modelview projects from Object space to world space. i.e. inPosition was in Objectspace but out will be in screen space.

Phong Reflection Model

Diffuse, Specular, Ambient ← the objects color, seems flat

unabh. von Kameraposition

↑ sun spot, depends on position of camera, light and object

Per-Pixel lighting vs Per-Vertex

↳ for every pixel computed, interpolates vertex normals

For the vertices computed and then only interpolated (faster, uglier, doesn't work if Beleuchtungsfläche < Fragment)

Geometry

Sphere: $(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2$

Optical Flow

$I(x,y,t) = I(x+u, y+v, t+1)$ ↤ Brightness Constancy

Smoothness Assumption ⇒ Taylor

$$I(x,y,t) \approx I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \underbrace{e}_{\approx 0}$$

Small timestep ⇒ $\frac{\partial I}{\partial x} = I_x$, $\frac{\partial x}{\partial t} = u$, $\frac{\partial I}{\partial y} = I_y$, v, $\frac{\partial I}{\partial t} = I_t$

Aperture Problem → (2 unknowns) $I_x \cdot u + I_y \cdot v + I_t \approx 0$

$\Leftrightarrow \nabla I \cdot \vec{u} \approx 0$

Normal Flow

perpendicular to above eq line.

$$u_\perp = -\frac{I_t}{|\nabla I|}\frac{\nabla I}{|\nabla I|}$$

Horn & Schunck

minimize $\underbrace{e_s + \lambda e_c}_{F}$, $e_s = \iint ((u_x^2 + u_y^2) + (v_x^2 + v_y^2))\,dx\,dy$

$e_c = \iint (I_x u + I_y v + I_t)^2\,dx\,dy$

⇒ Euler-Lagrange eq $F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} = 0$

⇒ $\Delta u = \lambda (I_x u + I_y v + I_t) I_x$

⇒ $\frac{\partial u}{\partial t} = \Delta u - \lambda (I_x u + I_y v + I_t) I_x$

$\frac{\partial v}{\partial t} = \Delta v - \lambda (I_x u + I_y v + I_t) I_y$

// More than 2 frames allows better estimate of I_t

// Information spreads from corner-type patterns

// Errors at boundaries

Lucas-Kanade

Spatial Coherence Constraint: Assume neighbours move with the same Displacement

$0 = I_t \cdot \nabla I(p_i) \cdot [u,v]$

⇒ more eq per pixel

⇒ least squares

$Ax = b \Rightarrow A^T A x = A^T b$

$\Rightarrow x = (A^T A)^{-1} A^T b$

(Also, temporal persistence (smooth moving) and Brightness constancy)

Lucas-Kanade contd (from the right)

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_n) \end{bmatrix}$$

Assuming a single velocity for all pixels within an image patch

$E(u,v) = \sum_{x,y \in \Omega} (I_x(x,y) u + I_y(x,y) v + I_t)^2$

Least Squares:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{(A^T A)} \begin{bmatrix} u \\ v \end{bmatrix} = -\underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{(A^T b)}$$

To be solvable, $A^T A$ should be invertible. The Eigenvalues of $A^T A$ should not be too small and their proportion $\frac{\lambda_1}{\lambda_2}$ (← Larger EW) should be small (well-conditioned)

⇒ not good in uniform regions or edges
(EW too small; one dominant direction)

⇒ good for high-texture or corners: Gradients have Different directions, large EW, large magnitudes.

Errors when:
- point does not move like its neighbours
- motion is larger than 1px
- Brightness constancy not holding

Solutions:
- motion segmentation
- iterative refinement, coarse to fine, feature matching
- neighbourhood search with normalized correlation

Iterative refinement

Estimate velocity at each pixel (Lucas-Kanade)
Warp one image with this flow towards the other
Refine estimate by applying again.

Filtering

High Pass Filter $= -$ (Lowpass Filter) — Blurs, removes high frequencies

Bandpass = conv2 (lpf, hpf) //with different ~~thresholds~~ standard Deviation

Partial Derivative in x direction : $I * \frac{1}{2}[-1\ 1]$

Simple Smoothing : $\frac{1}{2}[1\ 1]$

Convolution = Correlation with filter turned by 180° (or flipped in both directions). That's no difference for symmetric like Gaussian.
Correlation is not associative

Smooth image and then take its derivative == $F*(\text{Gaussian} * \text{Image}) = (F*G)*I$

DoG ← can be precomputed
=> only need one convolution per image at runtime

Covariance Matrix

Eric Mink

$\mu = \frac{1}{q}\sum_{i=1}^{q} x_i \qquad \Sigma = \frac{1}{q-1}\sum_{i=1}^{q}(x_i-\mu)(x_i-\mu)^T$

A = Image minus mean
=> $A \cdot A^T = \Sigma$ if x_i is a column vector that contains one image.

somehow in Ex5 only $\frac{(\text{Img}-\text{mean})\cdot(\text{Img}-\text{mean})'}{\#\text{samples}\cdot(\#\text{samples}-1)}$ — statistics. Or $\Sigma = \frac{1}{\#\text{Images}} \cdot \sum_{i=1}^{\#\text{Images}} (x_i - \mu)^2$ // #images == #samples

PCA Compression

$\Sigma_{i,j} = E[(x_i-\mu_i)(x_j-\mu_j)]$

PCA uses EigenVectors of Cov-Matrix Σ

$\cancel{(\Sigma_{i,j} = E[(X_i - \mu_i)^{2}])}$, $\mu_i = E(X_i)$ = mean of pixel

Image => vector $X \cdot Y \times 2$ dimensional.

"SSD" means Sum of Square Differences

SSD: `X = A(:) - RePr(:)';` // ' is ' in Maths
`err = X' * X`

//Might make sense to discard largest
//Eigenvectors because they're mostly lighting

```
% Mean(:) stores the mean of faces
% V       stores the EV
[V,D] = eig(Σ)                                    % Largest EV
affinesubspace(:, 1:spacedimension) = V(:, 1:spacedimension)
A = imread( ~~~ )
A = imresize(A, IMG_SCALE, 'bicubic')
% Project
Pr = double(A) - Mean  % center around Mean => smaller numbers
localcoords = affinesubspace' * Pr(:)
% localcoord is spacedimension high
% To display, project back
RePr = reshape(((affinesubspace*localcoords) + Mean(:)),
               size(A,1), size(A,2))
```

Fourier

Transformed $f = (\mathcal{F}f)(y) = \int_{R^n} f(x)\, e^{-i\pi y x}\, dx$

oder $(\mathcal{F}f)(y) = \frac{1}{(2\pi)^{n/2}} \int_{R^n} f(x)\, e^{-iyx}\, dx$

invFourier $= \mathcal{F}^{-1}(\mathcal{F}(f))(x) = f(x) = \frac{1}{(2\pi)^{n/2}} \int e^{iyx}\, \mathcal{F}(f)(y)\, dy$

convolution : $\mathcal{F}(f*g) = (2\pi)^{\frac{n}{2}} \mathcal{F}(f) \cdot \mathcal{F}(g)$

$\mathcal{F}(f) * \mathcal{F}(g) = (2\pi)^{\frac{n}{2}} \mathcal{F}(f \cdot g)$

Dirac Delta

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases}$$

To get rid of a signal, fourier transform it to find the frequencies, then bandpass (or lowpass) filter it. e.g. the sinus funfact can be removed by removing (u_0, v_0)

$\text{fourier}(\delta(x)) = \hat{\delta}(y) = \int_{-\infty}^{\infty} e^{-2\pi i x y}\, \delta(x)\, dx = 1$ because that is the sine frequency

=> $\text{invFFT}(f(y)=1) = \delta(x)$

holds also in 2D:

$$\iint e^{-2\pi i (ax+by)}\, dx\,dy = \delta(a,b)$$

Sinus Funfact

$$\sin(2\pi u_0 x + v_0 2\pi y) = \left(e^{2\pi i (u_0 x + v_0 y)} - e^{-2\pi i (u_0 x + v_0 y)}\right) \cdot \frac{1}{2i}$$

Sinus Aufbau

$y(t) = \sin(2\pi \cdot \text{freq} \cdot t + \varphi) \cdot A$ — frequency, phase, amplitude

Perspective Projection

$P'_x = \frac{P_x}{-P_z}, \quad P'_y = \frac{P_y}{-P_z}$ => $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -1 & 1 \end{bmatrix}$

camera space

to scale z and w to 1

to transform into camera space

Pyramid Sampling

Every level contains a blurred and sampled at every second pixel image (gauss Pyramid) or the difference to the next smaller image (Laplacian).

Recovering Function from samples

$x(t) = \sum_n x(nT) \cdot g(t - nT)$

$g(t) = \frac{\sin(\pi f_s t)}{\pi f_s t}$

Video Flow: Frame Types

I-frame: Intracoded, independent of other frames
P-frame: based on previous I & P frames
B-frame: both previous and future frames encode this

Temporal redundancy reduction is ineffective if there's much motion or scene change => partition frame into blocks and describe each block by finding the best matching one in the reference frame ("block matching")

Mahalanobis Distance

Measures Distance of point to set. If every axis has been scaled to unit variance, this == Euklid. distance

$D_M(\vec{x}) = \sqrt{(\vec{x}-\vec{\mu})^T \cdot S \cdot (\vec{x}-\vec{\mu})}$

$S_{i,j}$ = denoted $\Sigma_{i,j} = E[(X_i - \mu_i)(X_j - \mu_j)]$ = mean = Sum divided by #samples

Sobel

Gradient Magnitude $= \sqrt{Grad_x^2 + Grad_y^2}$

(Direction) Angle of Edge $= \arctan\left(\frac{G_y}{G_x}\right)$ // first move colors to $[-1,1]$ space

$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ (Prewitt $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$)

From solutions:
1. smooth w/ gaussian
2. Compute gradient magnitude & angle
3. Nonmaxima suppression
4. Double thresholding
5. Reject weak edge pixels that are not connected with strong.

Canny Edge Detection

0. Blur to filter noise
1. Detect edges and thin them by only keeping maxima. (Considering direction of Gradient)
2. Something between two threshholds is an edge if it is connected to an edge. Anything > both is an edge, anything below is not.

Combine Blur and sobel because differentiation is a convolution and convolutions are associative. So choose Gaussian filter and then

$D*(G*Img) = (D*G)*Img$

Harris Corner Detection

$\begin{bmatrix} \Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y^2 \end{bmatrix} \rightarrow$ Take Eigenvalues. Both large => Corner

⊕ Rotation invariant
⊕ Shift invariant
⊖ not scale invariant

Corner Detection

Maximize both Eigenvalues => Variance in every direction => Corner

SIFT

Blur image on different scales to get scale images. Use successive scale images, by subtracting them, to get an approximated Laplacian. This is scale-invariant: a factor σ disappears, assuming the scales were close together. Find the max/mins (Laplacian == 0). Get rid of found keypoints with low contrast or which are not in a corner.

Store orientation(s) of neighborhood by dumping copies of the keypoint into orientation buckets (neighborhood window = size of gauss blur filter)

Fingerprint keypoints by looking at 16 4x4 neighbours orientations => 128 number fingerprint. normalize and rotate it => rotation, shift, Illumination, scale invariant.

↑ because orientation is capped at 0.2 for fingerprint

Morphing Semantics

$I_1 \cup I_2$, $I_1 \cap I_2$ as expected

I^c = Complement. $= \{x : x \notin I^c\}$

$I_1 \setminus I_2 = \{x : x \in I_1 \text{ and } x \notin I_2\}$

ϕ = Empty Set

S fits I if the stamps' 1's match only 1's (at least one)

S hits I if the stamps' 1's match at least one 1

S misses I if the stamp's 1's doesn't match any 1

$\boxed{\text{S is called "Structured Element"}}$

Erosion $E(x) = I \ominus S = \begin{cases} 1 & \text{if S fits I at } x \\ 0 & \text{otherwise} \end{cases}$ => usually gets rid of border pixels (makes them black)

Dilation $D(x) = I \oplus S = \begin{cases} 1 & \text{if S hits I at } x \\ 0 & \text{otherwise} \end{cases}$ => more noise outside of body, less inside. (Gets rid of everything except a border if we do $(A \oplus S) \cap A^c$)

"opening" of I by S $= I \circ S = (I \ominus S) \oplus S$ = Dilation of Erosion (Removes small objects and kinda restores the shape)

"closing" $= I \bullet S = (I \oplus S) \ominus$ Erosion of Dilation (Get rid of pepper noise within Object)