Approximate Nearest Neighbors

Wegen der Curse of Dimensionality und der unteren Schranke von $\Omega(n^{1-1/d})$ für orthog. Range Queries glauben viele, daß es keinen Sinn macht, Algos für NN in hohen Dimen zu suchen, die (signifikant) besser als $O(n)$ sind (und nur Platz $O(n)$ brauchen).

Was kann man tun? Mehr Platz spendieren: etwas machen. Nicht exakten NN suchen (reicht für die allermeisten Appl.)

Dickerson, Duncan, Goodrich: Bd-trees are better when cut on the longest side, ESA 2000.
Duncan, Goodrich, Kobourov: Balanced aspect ratio trees ...
Arya, Mount, Netanyahu, Silverman, Wu: An optimal alg. for approx nearest neighbor searching –, 1998, S. 18+19

Def.:
Sei $S \subseteq \mathbb{R}^d$ eine Menge von Pkten, $q \in \mathbb{R}^d$ ein Query-Pkt, $p^* \in S$ der NN, und $\varepsilon > 0$.
Dann heißt ein $p' \in S$ "$(1+\varepsilon)$-approximate nearest neighbor" $\Leftrightarrow$

$$d(p', q) \leq (1+\varepsilon) \cdot d(p^*, q) .$$

Bezeichnungen:
Ab jetzt: v und $R(v)$ bzw. "Knoten" und "Region des Knoten" sind Synonyme.
Annahme: metric space, d.h. $d()$ ist Metrik, z.B. $d(p,q) := \|p-q\|$, $d(v,q) := d(R(v), q)$

$p_v :=$ der in v liegende Pkt, falls v Blatt ist
(sonst hatten wir ja $P(v) =$ Menge der in v liegenden Pkte)

Dadurch, daß wir mit einem "unendlich weit entfernten" Pkt starten und bei rast, geht der Weg zunächst direkt zu dem Blatt, das am dichtesten an q dran ist, oder wo q drin liegt. Von da aus geht er sukzessive zu immer weiter entfernt liegenden Blättern.

Algo:

Q = Priority-Queue mit Zeiger auf Knoten im k-d-Tree, sortiert nach $d(v,q)$ = Abstand zw. q und v; (kürzester Abstand vorne) (kurz am Anfang)

p° := Pkt innerhalb weit weg (= aktueller Kandidat des NN)

v := Root des k-d-Trees

(Achtung: es ist sehr wichtig, dass man hier mit einem "endlichen" Pkt startet! sonst könnte es passieren, dass die while-Schleife erst gar nicht startet!)

<u>while</u> $d(v,q) < \frac{1}{1+\varepsilon} d(p^\circ, q)$:

  <u>while</u> $v \neq$ Blatt : (v is inner node)

    seien v_1, v_2 die Kinder von v, und $d(v_1,q) \leq d(v_2,q)$

    füge v_2 in Q ein

  <u>endwhile</u>

  $v := v_1$

  <u>if</u> $d(p^v, q) < d(p^\circ, q)$: { v ist jetzt Blatt }

  <u>then</u>

    $p^\circ := p^v$

  $v \leftarrow$ extractMin(Q)

  $v :=$ minimales Element aus Q, lösche dieses aus Q;

<u>endwhile</u>

return p°

(Das entscheidende ist, dass der Algo das p^* gar nicht kennen muss! Ist für $\varepsilon=0$ genau der normale Algo, nur eben in iterativer Form)

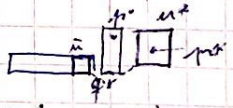
Zeit: $O(l \cdot \log n)$, l = # besuchter Blätter

Bew.:

1. Korrektheit: Sei u^* Blatt aus k-d-Tree, das p^* enthält

a) u^* wurde besucht $\Rightarrow$ Algo liefert $p^* \Leftrightarrow p^\circ$

b) u^* wurde nicht besucht $\Rightarrow p^\circ \neq p^* \Rightarrow$



$$d(p^*,q) \geq d(u^*,q) \geq d(\bar{u},q) \geq \frac{1}{1+\varepsilon} d(p^\circ,q)$$

(weil $d()$ Metrik; nähere Knoten werden zuerst besucht; Abbruchkrit.)

wobei $\bar{u}$ der zuletzt besuchte Knoten (Blatt oder innerer Knoten) sei.

2. Zeit:

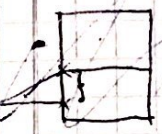
Pro Schleifendurchlauf: 1x Knoten extrahieren, $O(\log n)$ Knoten einfügen; (innerer loop: $O(\log n)$ mal)

Normal heap: $O(\log n)$ for extract & insert $\Rightarrow O(l \cdot \log^2 n)$

Fibonacci-Heap verwenden $\Rightarrow O(\log n)$ für Extrakt., $O(1)$ amortisierte Kosten für Einfügen; $d(v,q)$ braucht $O(1)$ $\Rightarrow$ Beh.

Bem.:

1. In der Praxis bleibt Heap relativ klein → verwende normalen Heap.
2. ~~$d(v,q)$ braucht nur $O(1)$,~~ nicht $O(d)$, weil man Distanz inkrementell aus Dist. zum Vater berechnen kann.
3. Analog geht "$(1-\varepsilon)$-farthest neighbor".



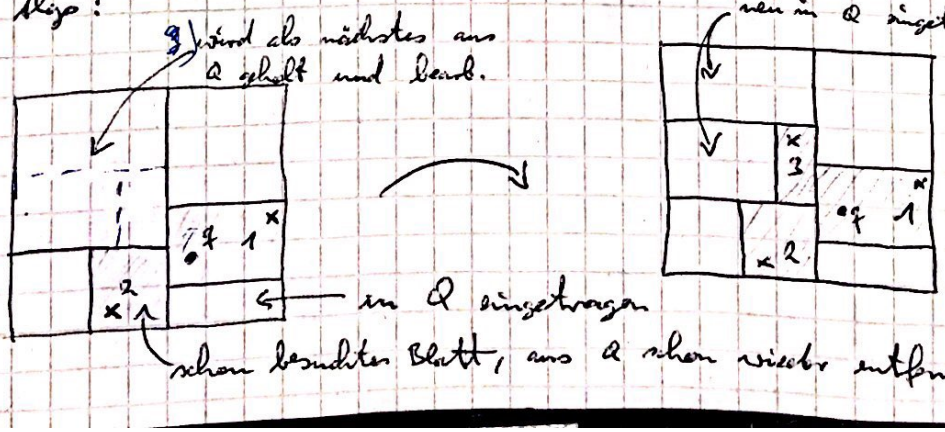
4. Gibt PD-Software "ANN Library", die gibt inzwischen bessere. S. Folien verwendet aber eine wesentlich komplexere Datenstruktur, weil damals noch nicht bekannt war, daß auch die alten "longest-side kd-Trees" diese schöne worst-case-Laufzeit haben.
5. # besuchter Blätter $l \in O(\log^{d-1} n)$ [ESA 2000]; zieht man, indem man Schranke für Anzahl der Knoten des kd-Tree findet, die einen Kreis um q "durchstechen". Genauer: $O\left(\left(\frac{\log n}{\varepsilon}\right)^{d-1}\right)$ gilt nur für "longest-side" kd-Tree!

Satz:

Zu einer Menge $S \subseteq \mathbb{R}^d$, $|S| = n$, und einem Anfr.-Pkt $q \in \mathbb{R}^d$ findet man den $(1+\varepsilon)$-nearest neighbor in Zeit $O\left(\frac{\log^d n}{\varepsilon^{d-1}}\right)$.

Bsp zum Algo:

kd-Tree, soweit er in der Queue eingetragen ist/war:



"Beste" ANN-Algos

Achtung: "beste" hängt von vielen Faktoren ab!
- Metrik oder nicht-metrischer Abstand
- Verteilung der Pkte (hohe Korrelation oder uniform)
- Dimension

Folgende zwei Algos funktionieren in praxi sehr gut für viele Daten

Alternatives Abbruchkriterium: besuche max L_{max} viele Blätter.

Alternatives Güte-Maß des ANN-Algo:

$$\text{precision} = \frac{\#\text{exakter NN's}}{\#\text{Queries}}$$

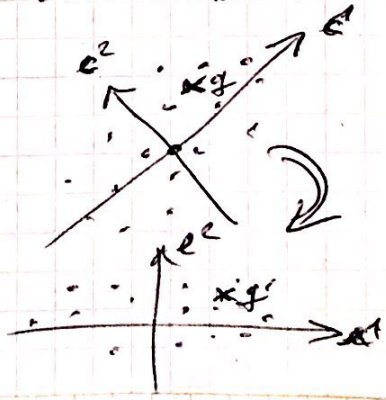
error = 1 − precision (eigtl. "error")

Randomized kd-Tree (RKD):
- Bestimme D Achsen (dimensions) mit höchster Varianz der Pkte
- Wähle eine davon zufällig
- Splitte entlang (across) Median dieser Achse

Experimente zeigen: D=5 reicht

[Muja & Lowe: Fast approximate Nearest Neighbors with automatic Algo configuration; 2009.
Silpa-Anan & Hartley: Optimised KD-trees for fast image descriptor matching; 2008.]

PCA-RKD:
- Bestimme Principal Components der Pkte P
- Transformiere Pkte → P'
- Baue RKD über P'
- Transformiere Query-Pkt q → q'
- Weiter mit bisherigem ANN-Algo.



RKD-Forest (auch mit PCA): Analog Rand. Forest

Baue mehrere (20-50) RKD-Trees über P (bzw P')

ANN-Search mittels RKD-Forest:

(Erinnert ein wenig an Random Forests!)

Maintain 1 p-queue only (one for all)

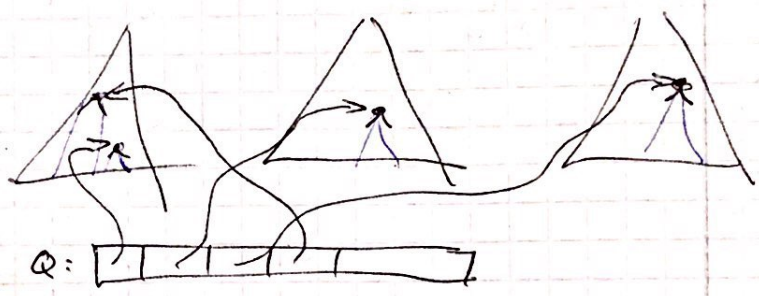
Descend each RKD-tree down to leaf closest to q (first iteration of outer while loop of ANN algo)

Choose closest p of all those leaves

put others in p-queue

Proceed with ANN-algo as before

Queue contains pointers to nodes in different trees!



Warum funktioniert das besser?

Bsp.: 1 Mio Pkte $\approx 2^{20}$ ⇒ Tiefe des kd-Trees = 20

Dimension = 100 ⇒ normaler ANN schaut 80 Einträge von q gar nicht an!

⇒ q und p' in Blatt sind nahe (close) zueinander in 20 Dim., aber in den 80 anderen nicht notwendigerweise

Hier hilft die Randomisierung und Ensemble

andere Betrachtung:

- Beim klassischen ANN-Algo hängen die nacheinander (aus der P-Queue kommenden) besuchten Zellen stark voneinander ab
- Bei obigem Algo liegen die Zellen aus der P-Queue in verschiedenen, randomisierten kd-Trees ⇒ weniger Abhängigkeit dazwischen!

k-Means-Tree:

1. Partitioniere P mittels k-means-Clustering in k Cluster
2. Erzeuge für jeden Cluster einen k-Means-Tree
3. Erzeuge Knoten mit k Kindern

ANN-Search mittels k-Means-Tree:

Traversiere Baum bis zum nächsten Blatt
Schreibe nicht-besuchte Kinder in P-Queue
Sortiere P-Queue bzgl. dist(q, cluster-center)
Stoppe nach Lmax besuchten Blättern

Bem.:

- k-means-Clustering benötigt auch NN (allerdings immer nur zu den k Mittelpunkten der Cluster)
- Randomisierung und Forest haben hier nichts gebracht (lt. Autoren)
- Aufbau ist rel. teuer → führe nur wenige Iterationen in k-means-Clustering durch, nicht bis Konvergenz laufen lassen. ANN wird kaum beeinträchtigt

Evtl. noch kNN auf GPU?

Folien mit "Results" zeigen

Anwendungsbsp. 2: Textursynthese

("texture synthesis", "image inpainting") [Wei & Levoy: Fast Texture Synth Siggraph 2000]

Gegeben: ~~Textur~~ ~~Image~~ I

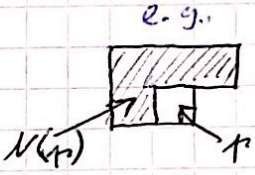
Gesucht: größere Textur T, die genauso aussieht

Beob.:

1. ~~Der Ausschnitt in einem mit dem~~ "moving window" sieht immer gleich aus.
   (muß hinreichend groß sein)
2. Jedes Pixel hängt nur von einer gewissen lokalen Nachbarschaft ab

(Beides ist für "normale" Bilder nicht erfüllt.)

Bezeichnung: p_i = Input-Pixel aus I, p_o = Output-Pixel aus T,
$N(p)$ = Nachbarschaft von p (p is not member of $N(p)$!)



Algo:

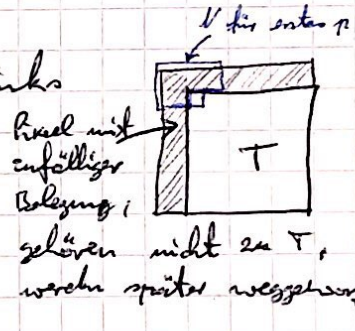
init T := schwarz plus Zufallsrand aber v. links

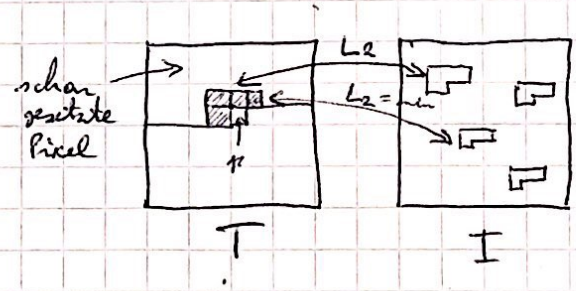
for all $p_o \in T$ in scan order:

(*) suche derjenige $p_i \in I$, mit

$$\| N(p_o) - N(p_i) \| = \min$$

setze $p_o := p_i$





Schritt (*) ist eine Nearest-Neighbor-Suche!

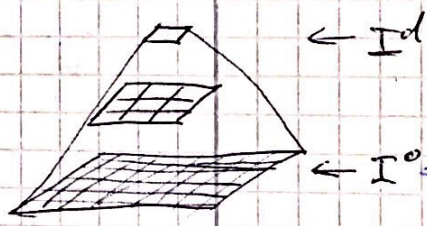
$N(p)$ sind d-dim. Vekte, Distanzat ist eine Metrik (d = # pixels in $N(p)$)

→ verwende k-d-Trees.

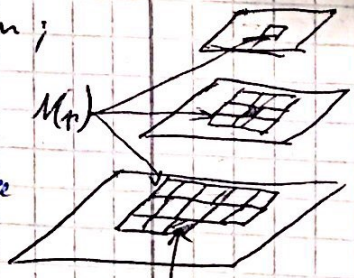
Ränder:
- ~~Betrachte von I~~ ~~klebe I zu T~~ nur die inneren $N(p_i)$. (setzt voraus, daß I groß genug ist, so daß Randpixel so ähnlich nochmal im Inneren vorkommen)
- Wrap-around für rechten Rand in T (braucht man nicht für linken Rand).

Bem.:
- Alle Pixel hängen deterministisch von den zufälligen Pixeln am linken u. oberen Rand ab (und I).
- (*) Die Qualität hängt von der Größe von $N(p)$ ab (s.u.);
- die Form von $N(p)$ ist beliebig (any), muß nur "kausal" sein.
- Andere "Scans" sind möglich u. sinnvoll, z.B. spiralförmig, um ein Loch zu "stopfen"

(*) Log. für das Problem der "richtigen" Größe der Nachbarschaft:
Image-Pyramide - unterster Level = orig. Bild; jeder nächsthöhere Level entsteht (gebildet) aus dem darunter durch Filterung (z.B. Gauss-Filter) und Subsampling.



Hier:
baue Pyramide $I^0, I^1, \dots, I^d$ aus I; (bottom-up) (used for T based pyramid top-down)
erzeuge T^l aus $I^l, I^{l+1}, I^{l+2}, \dots, I^{l+k}$ wie oben;
Nachbarschaft $N(p)$ für $p \in T^l$ erstreckt sich über $T^l, \dots, T^{l+k}$;
~~baue für jeden Level~~ for each level l, build kd-tree I^l einen kd-Tree mit Ebten, die aus der erweiterten Nachbarschaft entstehen, ~~d.h. aus $I^l, \dots, I^{l+k}$~~
baue für jeden Level l einen kd-Tree mit Ebten $N(p_i) \subseteq I^l \cup \dots \cup I^{l+k}$.



↑ etwas salopp hingeschrieben, denn man müßte zuerst definieren, was ∪ hier bedeutet

Beschleunigung durch "Jump Maps":

[Zelinka & Garland: Towards Real-Time Texture Synth ... EG WS on Rendering 2002]

Idee: Nearest-Neighbor-Suche als Preprocessing

Im Input-Bild (Analyse):

Für jedes Pixel $P_i \in I$ im Input-Bild bestimme

k Nearest-Neighbors P_{ij} gelbst Wahrscheinlichkeit $p_{ij} = f\left(\frac{1}{d_{ij}}\right)$

basierend auf Dist $d_{ij} = d(P_i, P_{ij})$ → Jump Map

(große Dist ≙ kleiner W.keit)

Synthese-Algo:

Output-Pixel $P_o \in T$ in Scanline-Order (z.B.) durchgehen:

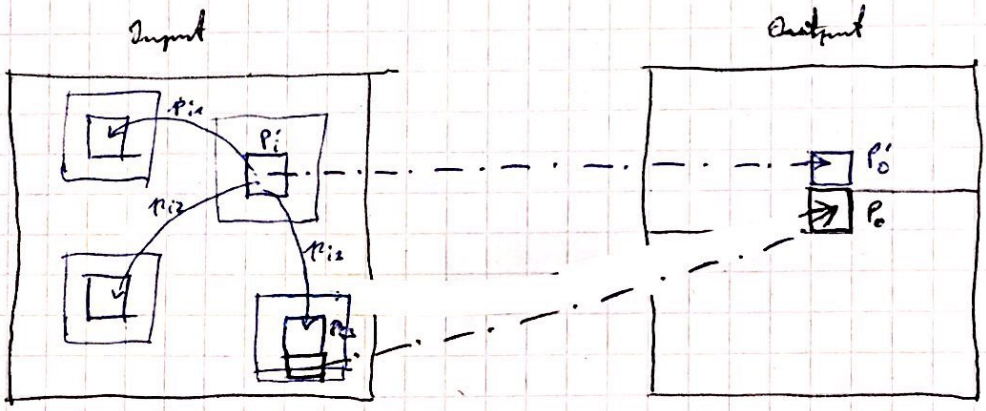
wähle zufällig Pixel $P'_o \in N(P_o)$ das schon berech. ist

sei $P'_i \in I$ Pixel aus Inputbild, aus dem P'_o ber. wurde

wähle zufällig, gemäß p'_{ij} mit W.keit, einen NN $P'_{ij} \in I$ von P'_i aus Jump Map

setze P_o := korrespondierendes analoges Pixel aus $N(P'_{ij})$

speichere dieses Herkunftspixel zu P_o



Analog mit Image-Pyramide:

Issues:

1. Diversifizierung der k Nearest-Neighbors sicherstellen;
   d.h., die NNs sollten hinreichend weit auseinanderliegen
2. Vermeidung von Ränderbildung:
   w.keit der
   erniedrige die Jumps, die in die Nähe des Randes
   des Input-Bildes springen
3. Synth.-Reihenfolge:
   besser bidirektionale Scanline-Ordnung;
   noch besser: entlang Hilbert-Kurve
   (dann kann man mit ca. gleicher W.keit
   Nachbarpixel aus allen 4 Richtungen wählen)

Resultat: ca. 800,000 Pixel/sec ≈
256×256-Bild in 0.1 Sek (Pentium 3, 1GHz)

Anwendungsbsp 1: <u>Shape-Matching</u>

Aufgabe: Suchen in Bilddatenbanken durch Bsp., Medizindatenbanken (Röntgenaufnahmen), Internet, Schrifterkennung.

Shape = 2D-Kurve, 3D-Oberfläche (z.B. aus Kantendetektor, Segmentierung, Polygone, ...)

Idee allg.:

1. Definiere Transf. Shape → "Deskriptor feature vector" ($\in \mathbb{R}^d$ i.A.)
   sollte invariant sein unter Rot. + Transl., Tesselierung, evtl. auch unter Skalierung + Spiegelung
   Deskriptor ist meist Vektor $\in \mathbb{R}^n$, manchmal auch Graph u.ä.
2. Definiere "dissimilarity measure" auf den Feature-Vektoren, so daß
   $d(f_1, f_2)$ groß $\Rightarrow$ S_1 und S_2 sehen sehr verschieden aus.
3. Baue zur Shape-Datenbank einen k-d-Tree auf (genauer: zu den Feature-Vektoren)
4. Suche best match zu Vorlage durch Nearest-Neighbor-Suche (evtl. k-NN)

Shape distribution:

[Osada, Funkhouser, Chazelle, Dobkin: "Matching 3D models with shape distributions" SMI 2001]

Wähle "shape function":

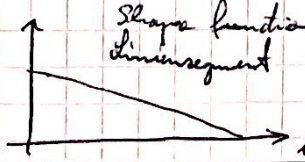
$$f(P_1, \dots, P_x) \to \mathbb{R}$$

z.B.: Abstand zw. Pkte, Flächeninhalt des Dreiecks dreier Pkte, ... geschätzte lokale Krümmung

Sample Oberfläche (=Rand) der Shape zufällig & uniform mit jeweils x Pkten → "shape distribution" (= feature vector)

Speichere als Histogramm.

Bsp. für $f(P_1, P_2) = |P_1 - P_2|$



Bsp.: Histogr. für Pkt's

Surflet-Pair Histograms:

[Wahl, Killenbrand, Hirzinger: "Surflet-Pair-Relation Histograms: A Statistical 3D-Shape Representation for Rapid Classification", 2003]

Betrachte wieder alle Paare p_1, p_2 Vertex- des Input-Meshes (oder Sample).

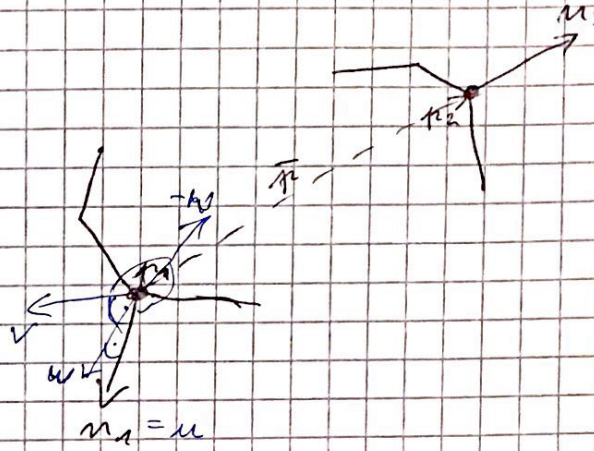
Seien n_1, n_2 die Normalen in p_1, p_2.

Setze $\bar{p} := \text{normalize}(p_2 - p_1)$

Wg Robustheit soll

$|n_1 \bar{p}| \leq |n_2 \bar{p}|$

sein, [also $\sphericalangle(n_1, \bar{p})$ "senkrechter" als $\sphericalangle(n_2, \bar{p})$]. Sonst p_1, p_2 vertauschen.



Konstruiere lokales Koord.system in p_1:

$u := n_1$

$v := \text{normalize}(\bar{p} \times u)$

$w := u \times v$

Beschreibe p_2 relativ zu p_1:

$\alpha = \arctan 2(w \cdot n_2, u \cdot n_2)$ ← (arctan mit V2) n_2 in die uw-Ebene projiziert und dann Winkel mit der w-Achse

$\beta = \arccos(v \cdot n_2) = \sphericalangle(v, n_2)$

$\gamma = \arctan 2(u \bar{p}, v \bar{p})$

$\delta = \arccos(w \cdot \bar{p})$

$\varepsilon = \|\bar{p}\|$

Quantifiziere $\alpha, \ldots, \varepsilon$ in z.B. $5^5 = 3125$ Bins

Damit Eigen: $f(p_1, p_2) \to [0,4]^5 \in \mathbb{N}^5$ = "feature vector"

Für geg. Geometrie, betrachte alle Paare (p_i, p_j),

bilde Histogramm aller Feature-Vektoren ⇒ Shape Histogram

Für DB von Shapes: berechne Shape-Histograms (preprocessing)

Note: invariant unter Rot. + Transl.!

Zur Query-Time suche MV zu einem gegebenen Histogramm h_Q.

Abstandsmaße: ("histogram similarity")

1. Euklidische Distanz
2. χ^2-Test
3. Likelihood: $L(O,Q) = L = \sum_{p_1, p_2 \in Q} \log h_O(\text{bin}(p_1, p_2))$

   wobei h_O = Histogramm eines Obj.es aus der DB
4. Kullback-Leibler-Divergenz:

$$K = \sum_{i \in \text{all bins}} h_Q(i) \cdot \ln \frac{h_Q(i)}{h_O(i)}$$

(asymmetrisches Maß für Unterschied zweier Verteilungen)

Ignoriere (oder bestrafe) Terme, wo $h_Q(i) \neq 0 = h_O(i)$.

Terme mit $h_Q(i) = 0$ sind OK, da $\lim_{x \to 0} x \ln(x) = 0$.

Eigenschaften: $K \geq 0$, $K = 0 \Leftrightarrow h_Q = h_O$.

K und L scheinen am besten zu funktionieren

(im Paper 99% recognition rate, waren aber nur 20 Obj.e)

Achtung: K ist keine Metrik!

(asymmetrisch, und Dreiecksungleichung gilt nicht!)

Eigenschaften: $K \geq 0$,

$K = 0 \Leftrightarrow h_Q = h_P$

Fast Iterative Closest Points (ICP):

Exkurs: Alignment / Registration of Shapes

[Eggert et al.: Estimating 3-D Rigid Body Transformations, A Comparison. 1997]

[Horn: Closed-form solution of absolute orientation using unit quaternions]

! Vorher Exkurs zu Quaternionen in Folien.pptx !

Gegeben: zwei Pkt-Mengen
$A = \{a_i\}$, $B = \{b_i\} \subseteq \mathbb{R}^3$.
Annahme: zwischen A, B besteht eine (perfekte) Korrespondenz der Form

$$a_i = R\,b_i + t \quad ,$$

wobei R = Rotationsmatrix, t = Translat. vektor
Gesucht: R, t

[Bemerkung: weil die Korrespondenzen später nicht perfekt sein werden, wollen wir hier nicht den naiven / straight-forward Ansatz wählen, bei dem man sich einfach die ersten 3 Pkte jew. nimmt, je ein lokales Koord. system für A bzw. B daraus baut, und diese dann ineinander überführt.]

Definiere den least squares - Fehler

$$E^2 = \sum_{i=1}^{n} (R\,b_i + t - a_i)^2$$

Setze $\bar{a} = \frac{1}{n}\sum a_i$, $\bar{b} = \frac{1}{n}\sum b_i$ (Schwerpkte)
Schreibe $a_i =: a_i' + \bar{a}$, $b_i =: b_i' + \bar{b}$

Damit: $E^2 = \sum_i \left(R(b_i + \bar{b}) + t - a_i - \bar{a} \right)^2$

$= \sum_i \left(Rb_i - a_i + R\bar{b} + t - \bar{a} \right)^2$

Wenn wir das optimale R kennen, dann gilt für das opt. t:

$Rb + t - a = 0 \quad \Leftrightarrow \quad t = \bar{a} - R\bar{b}$

Also definiere jetzt $E'^2 = \sum_i \left(Rb_i - a_i \right)^2$

Zur Vereinfachung: lasse Striche bei a_i, b_i ab jetzt weg.

Also $E^2 = \sum_i \left(Rb_i - a_i \right)^2$

$= \sum_i \left((Rb_i)^2 - 2(Rb_i)a_i + a_i^2 \right)$

$= \sum_i b_i^T \underbrace{R^T \cdot R}_{=I} b_i + a_i^2 - 2 a_i^T R b_i$

$= \sum_i b_i^2 + a_i^2 - 2 a_i^T R b_i \overset{!}{=} \min$

E^2 wird minimal $\Leftrightarrow \tilde{E}'^2 = \sum_i a_i^T \underbrace{R}_{3\times3\text{-Matrix}} b_i = \max$

Nebenrechnung (Magie) mittels Quaternionen:

$a^T R b = (0,a) \cdot q \odot (0,b) \odot q^*$ wobei q das Quaternion ist, das die Rotation R bewirkt

↑ Skalarprod! ↑ Quat.-mult!

stelle um

$= \left(q \odot (0\,b) \odot q^* \right)^T \begin{pmatrix} 0 \\ a \end{pmatrix}$

$= \left(Q^* (q \odot (0\,b)) \right)^T \begin{pmatrix} 0 \\ a \end{pmatrix}$ wobei Q^* die Matrix ist, die die Quat.-mult. mit q^* bewirkt (s. Folien-ppt)

$= \left(q \odot (0\,b) \right)^T \underbrace{Q^{*T}}_{=Q} \begin{pmatrix} 0 \\ a \end{pmatrix}$

$= \left(q \odot (0\,b) \right)^T \cdot \left((0\,a) \odot q \right)$

$= (Bq)^T \cdot (Aq)$ wobei A, B die Matrizen sind, die die Quaternionenmult. mit $(0\,a)$ bzw. $(0\,b)$ bewirkt

$= q^T B^T A\, q$

↳ Schreibe damit

$$E''^2 = \sum_i a_i^T R\, b_i = \sum_i q^T B_i^T A_i\, q$$

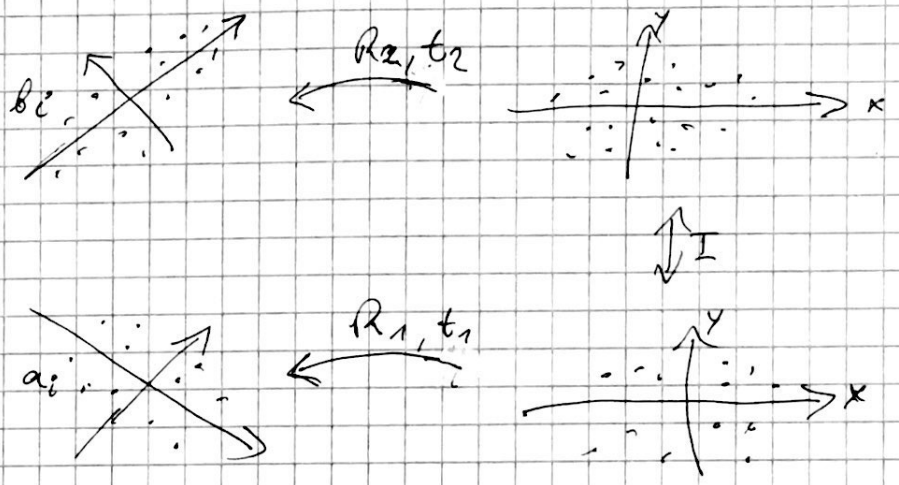
(a_i in q-repr.; Quaternion, das Rotation R repr.)

$$= q^T \underbrace{\left(\sum_i B_i^T A_i\right)}_{M} q \overset{!}{=} \max$$

$M = 4\times 4$-Matrix = Kovariana-Matrix (?)

Das Quaternion, das E''^2 max werden läßt, ist der größte Eigenvektor von M
⇒ transformiere diesen in Rot. matrix R → fertig

Oder: berechne PCA zu $\{a_i\} \to R_1$
" " " $\{b_i\} \Rightarrow R_2$
Rotation R von $\{b_i\}$ nach $\{a_i\}$ ist $R_1 \cdot R_2^{-1}$



$$a_i = R_1^{-1} \cdot R_2^{-1}\left(b_i - t_2\right) + t_1$$

*) Matrix enthält Terme der Form:
$-\sum_i a_x^i b_x^i + \sum a_y^i b_y^i + \sum a_z^i b_z^i$ * auf der Diagonale
$-\sum a_z^i b_y^i - a_y^i b_z^i$ auf Off-Diagonal-Elemente