Advanced Robot Perception

Fortgeschrittene Konzepte der Wahrnehmung für Robotersysteme

Georg von Wichert, Siemens Corporate Technology

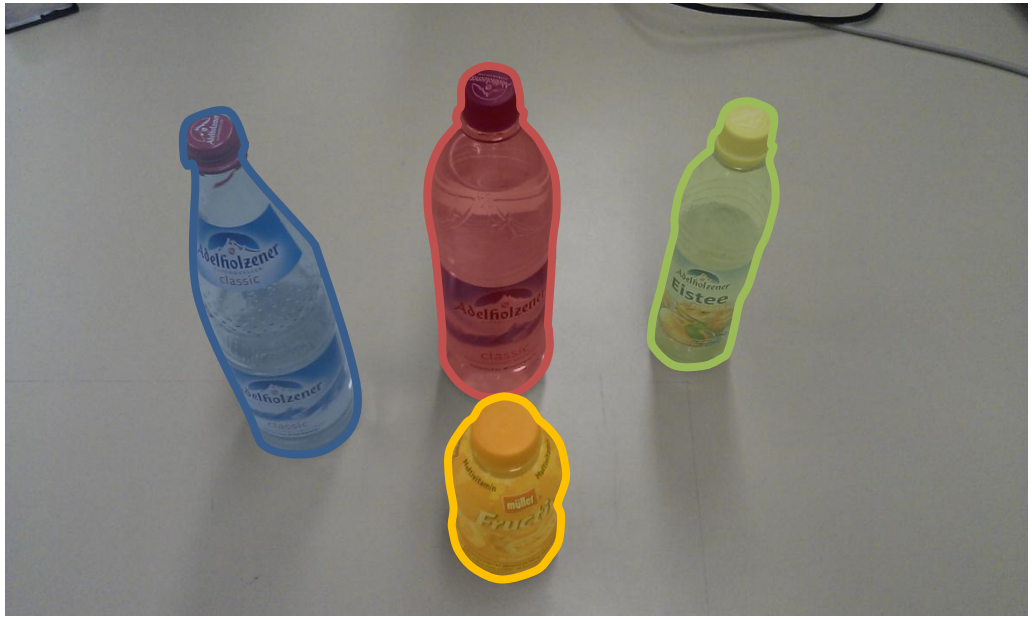
WAHRNEHMUNG MIT PUNKTEWOLKEN

Wahrnehmungsaufgabe

- Aufgabe: Extraktion von Fakten aus unstrukturierten Sensordaten, z.B.
  - Für den Roboter relevante Objekte in der Umgebung
  - Lage / Pose, Klasse, Instanz

Beispiel: Objekterkennung und -lokalisierung

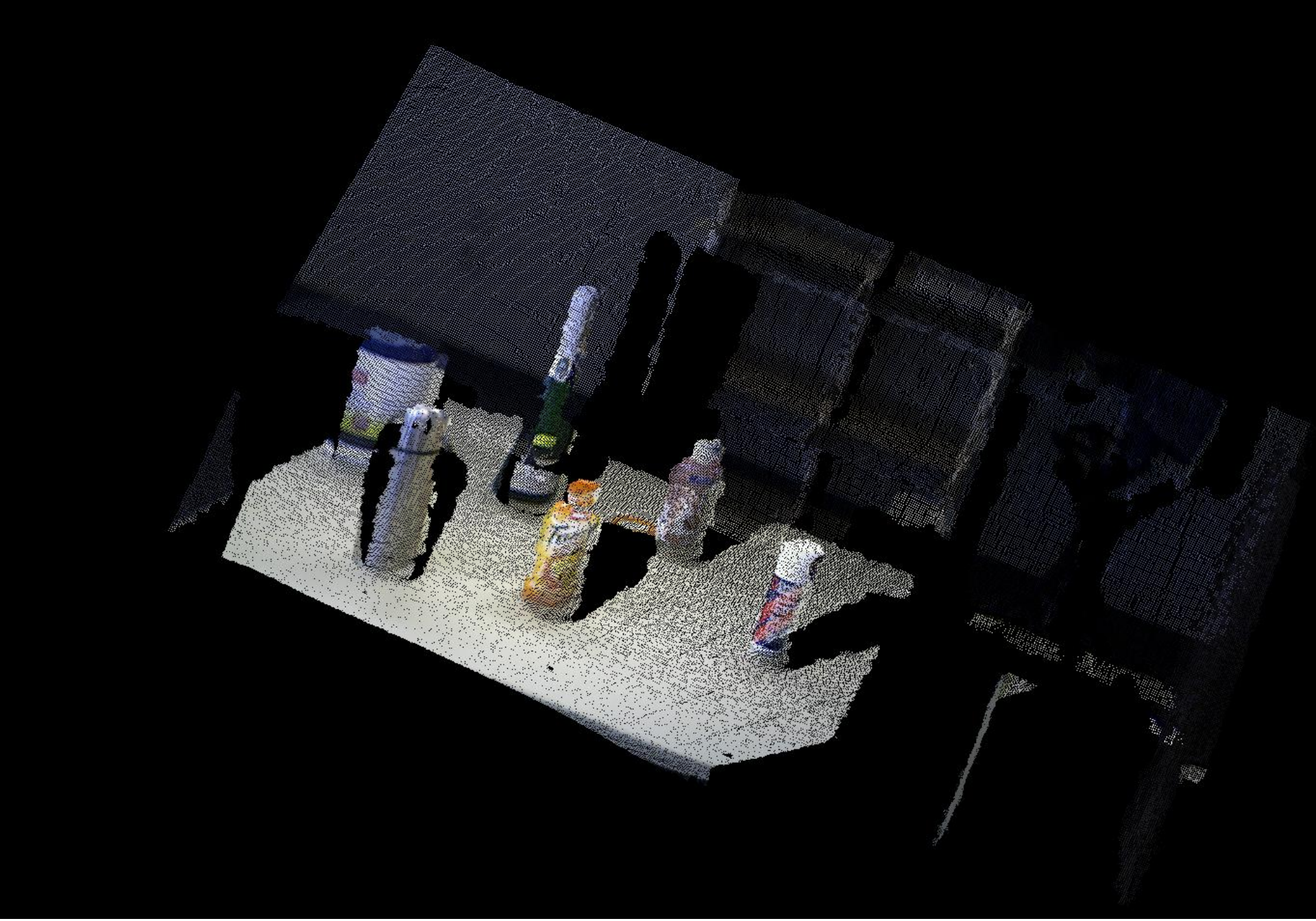
- Drei wesentliche Verarbeitungsschritte
  - Segmentierung
  - Gegebenenfalls Erkennung / Vermessung
  - Lagebestimmung

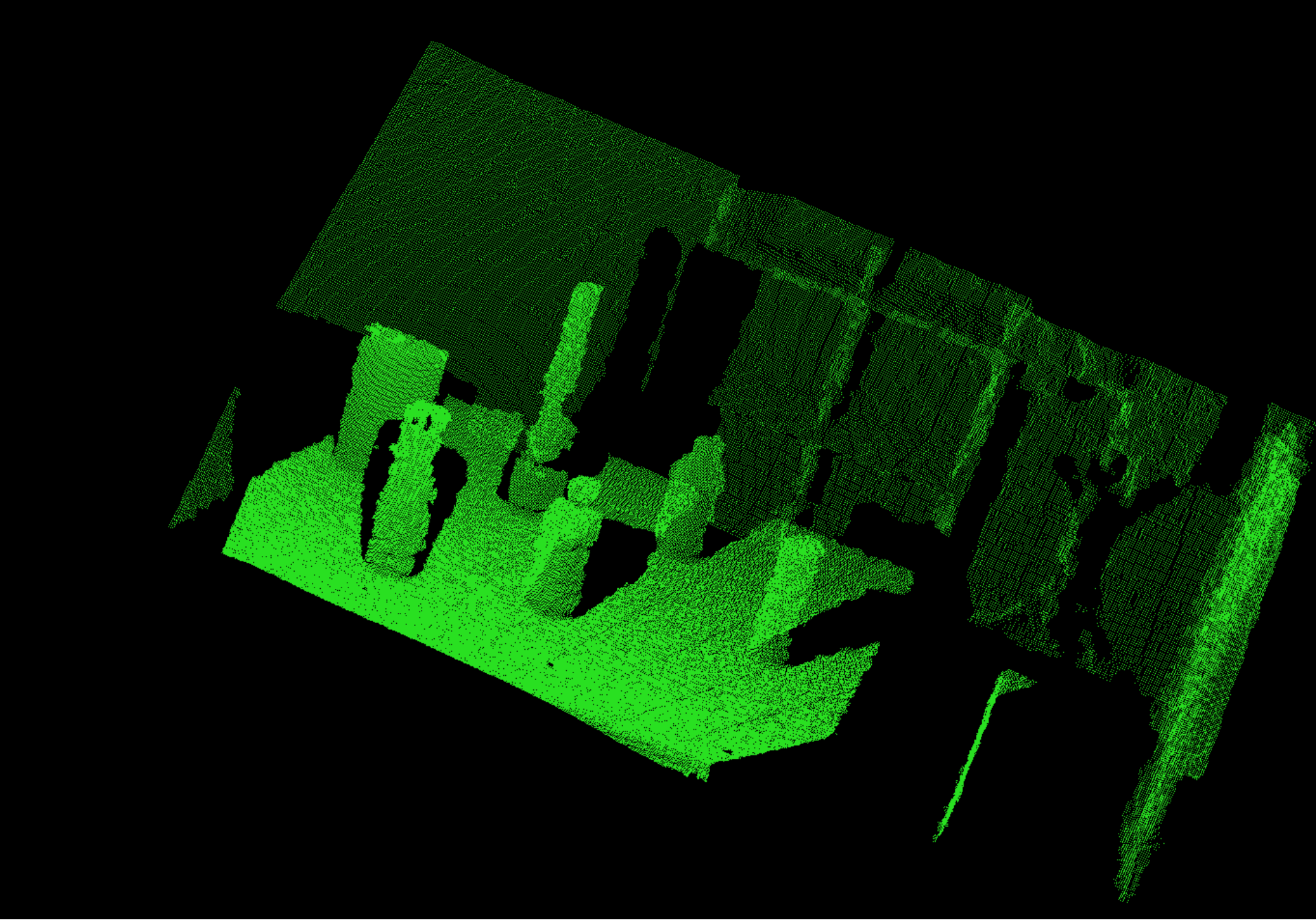


Segmentierung

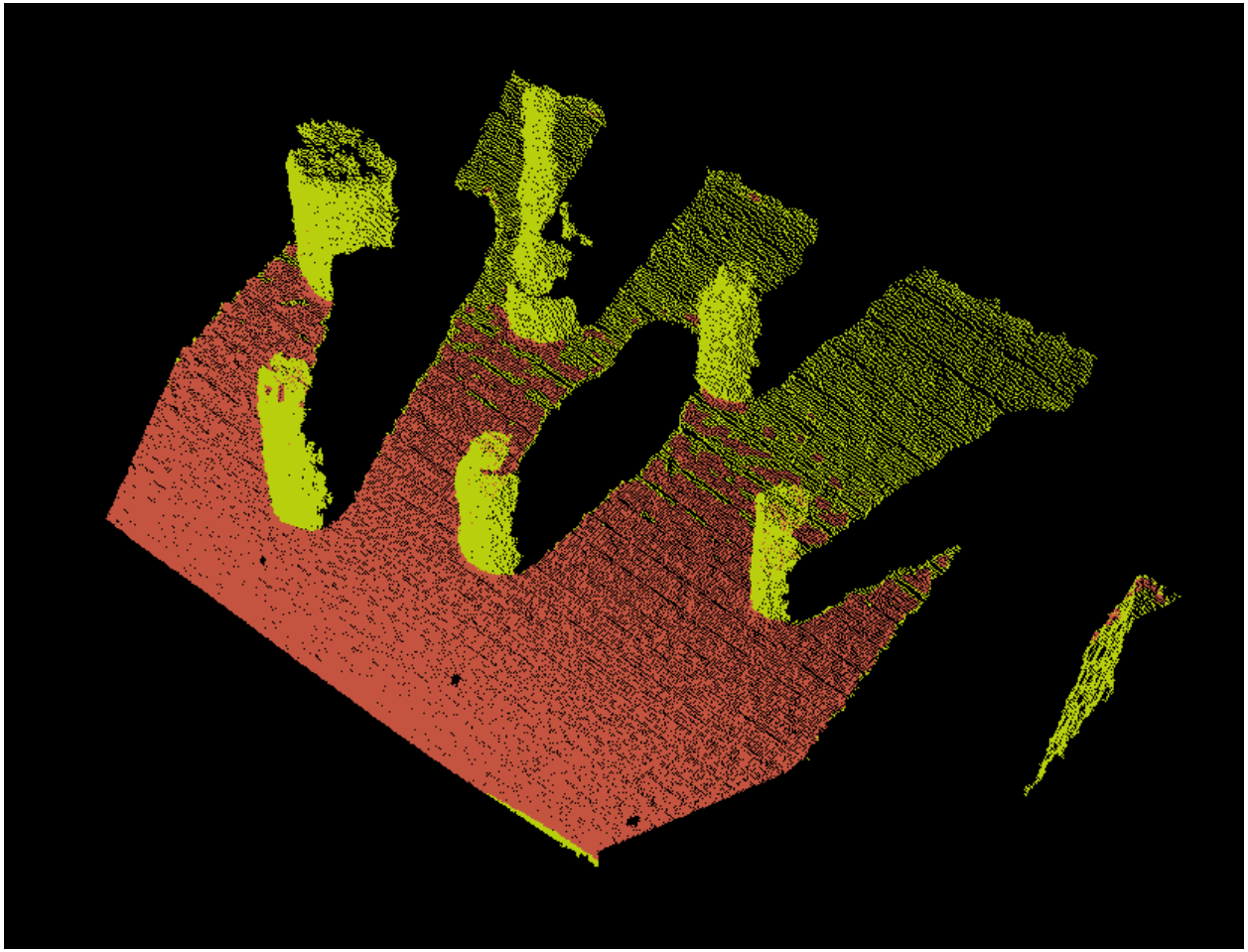
- Aufteilung der Daten in inhaltlich zusammenhängende Teilmengen
  - Was ist ein Segment? Kommt darauf an!
  - Segmente entsprechen beispielsweise Objekten
  - Definition (und Lösung) aufgabenabhängig

Binarisierung mit Schwellwert

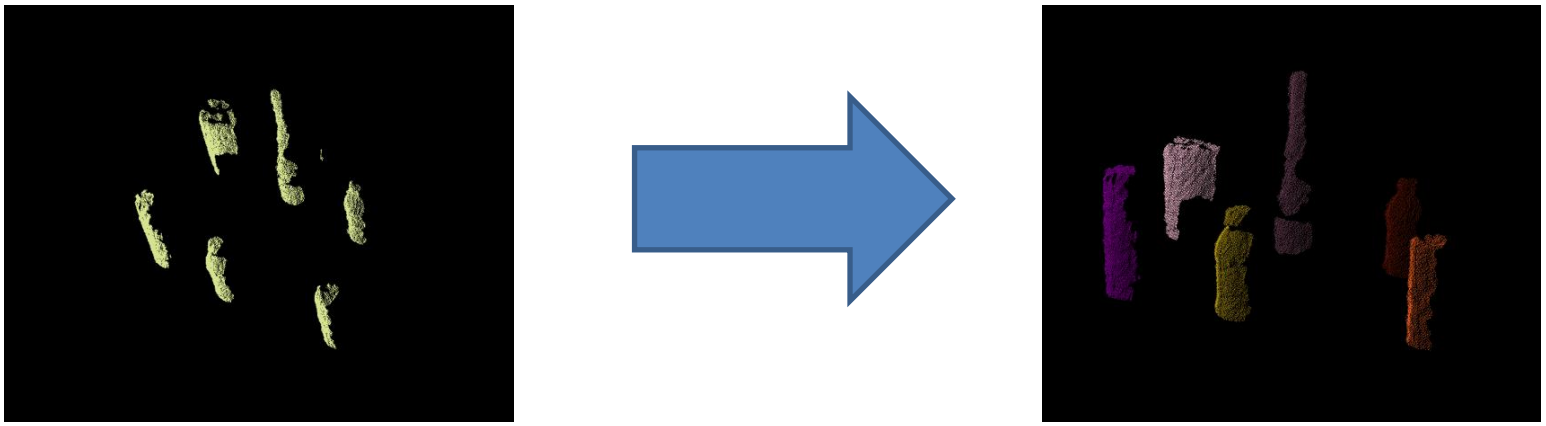




Ebenenfit mit RANSAC



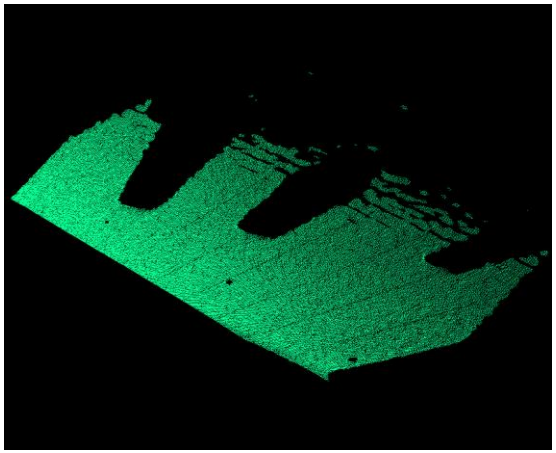
Extraktion der Einzelobjekte



- Pro Objekt gibt es eine zusammenhängende Teil-Punktewolke
  - Punkte, deren minimaler euklidischer Abstand untereinander kleiner als der zu den Nachbarobjekten ist
- Segmentierung mit dem sogenannten „Region growing“
  - Andere Möglichkeit z.B. k-Means

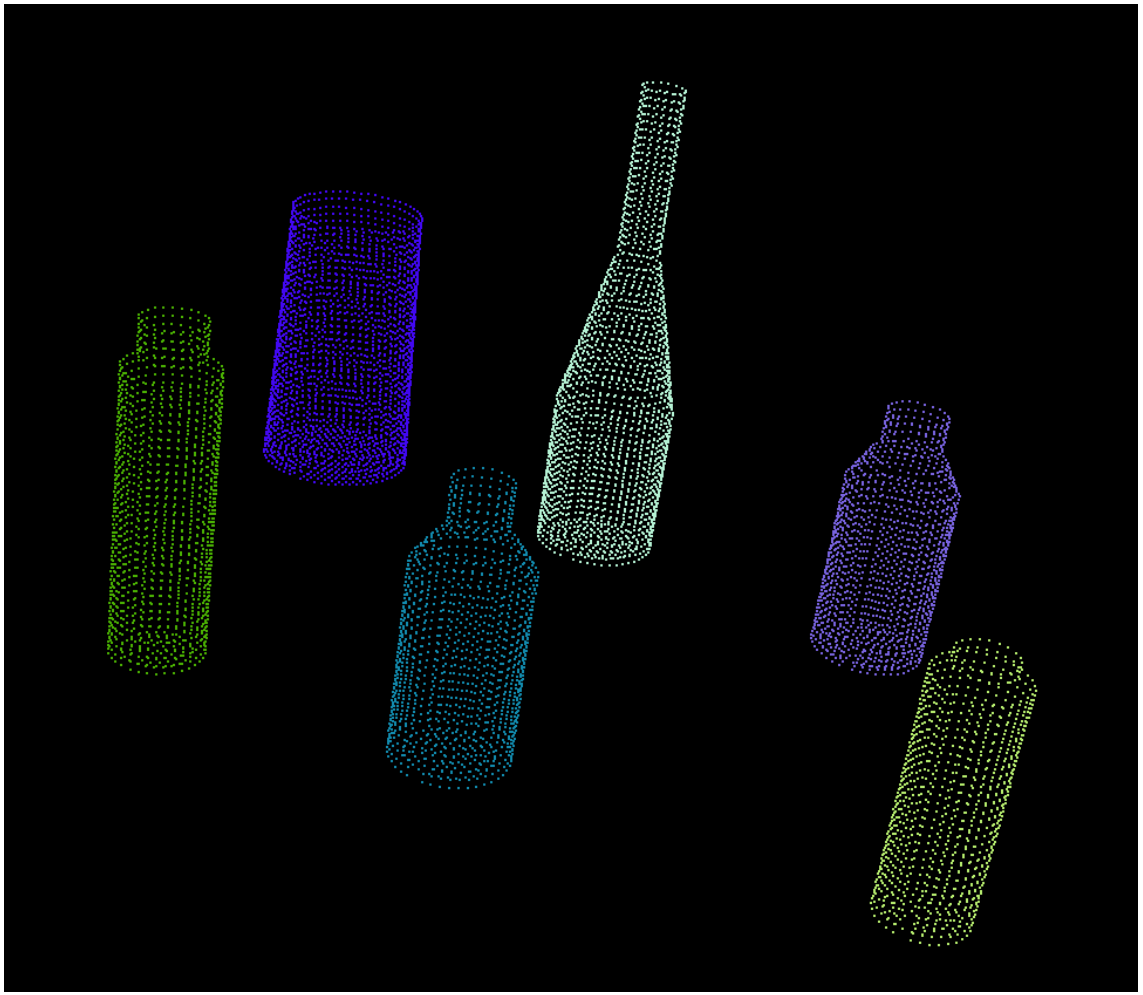
Einschub

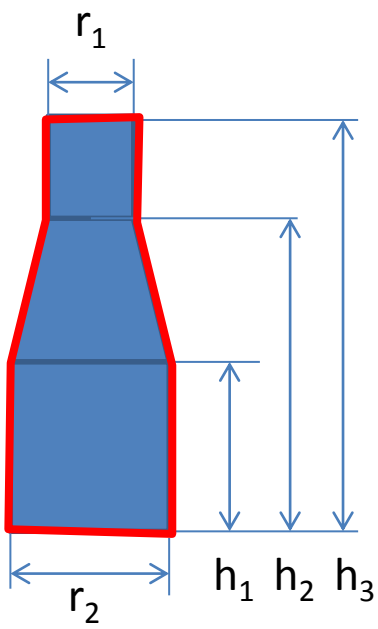
Segmentierung



- Segmentierung: Inhaltlich zusammengehörige Teilmengen der Daten finden
- Zwei grundsätzliche Ansätze
  - Segmentierung durch Modellanpassung („model fit“)
    - Ebenenfit mit RANSAC
  - Segmentierung durch Clustering
    - „Region Growing“
    - Clusteringverfahren aus der Statistik: Hierarchisches Clustering, K-Means, …

Vermessung und Lagebestimmung

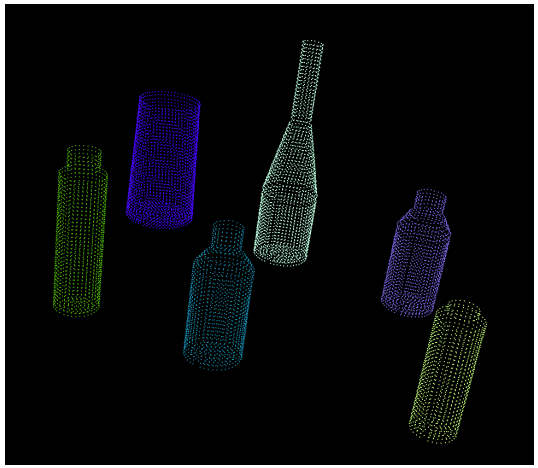




Was lernen wir jetzt daraus?

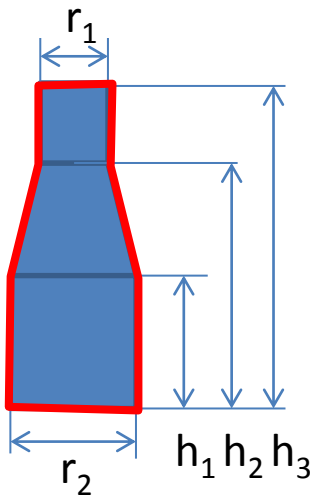
- Wahrnehmung: Extrahiert strukturierte Information aus unstrukturierten Sensordaten
  - Anzahl vorhandener Objekte
  - Position und Formparameter für jedes Objekt

- Das beschriebene Vorgehen ist nur eine von vielen Möglichkeiten, aber recht typisch
  - Modellannahmen: Einzelne Objekte auf Tischebene aufrechtstehend, rotationssymmetrisch, „flaschenförmig“

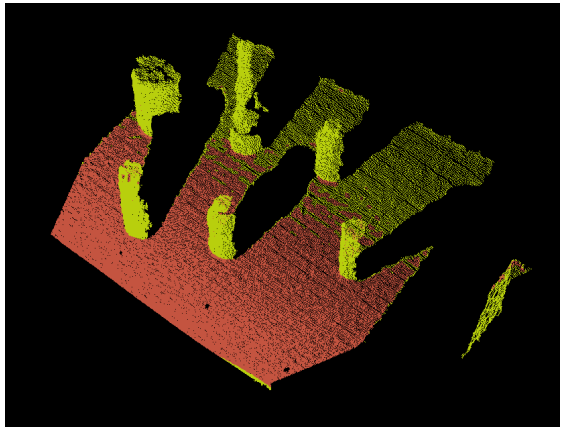


Was lernen wir jetzt daraus?

- Starke Dimensionsreduktion durch Wahl der Objekt- und Szenenmodellierung
  - Objektlage (senkrecht auf Tisch): 6D Pose -> 3D Pose
  - Rotationssymmetrie: 3D Pose -> 2D Pose
  - Abstrahiertes Flaschenmodell: Form 5D



- Schätzung der verbliebenen 7D Parametervektoren mit verhältnismäßig vielen Sensormessungen (3D Punkte) -> erhöht die Genauigkeit
  - Siehe auch Tischebene

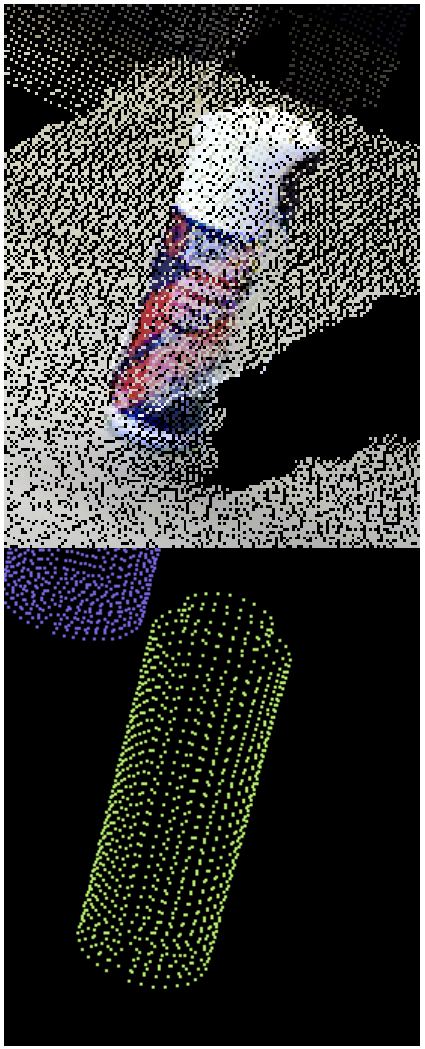


Was lernen wir jetzt daraus?

- Nutzt wissen über die Szene -> funktioniert nicht / schlecht bei Abweichungen von den Annahmen
  - z.B. sich (fast) berührende Flaschen stören die Segmentierung

- Aber auch wenn alles klappt: Fehler bleiben! Immer!
  - Messfehler: z.B. Objektradius … niemals exakt sein
  - Strukturfehler: Z… …ydose bekommt …
    - …
    - O… …(beim Zylinder üb… …meter für Minimierung des Ges…
    - Folge der Modellannahmen

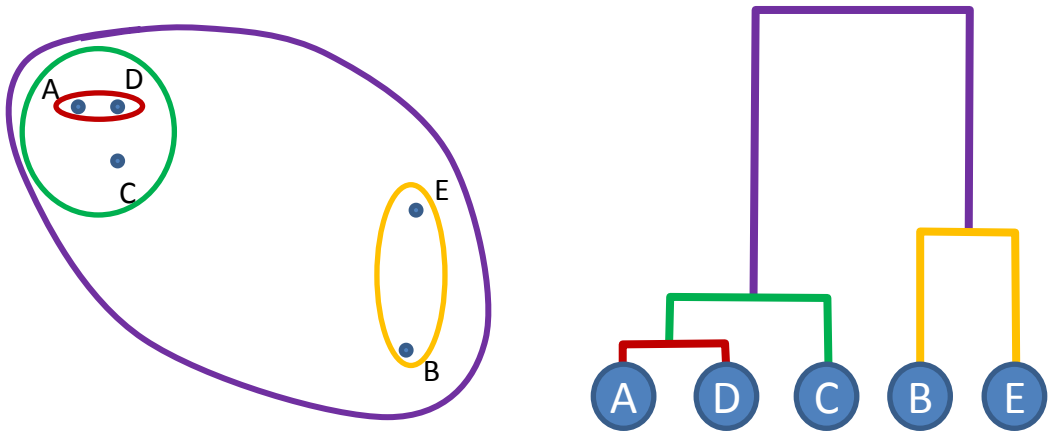
Es bleiben immer Fehler!!



Einschub

Hierarchisches Clustering

- Agglomerativ: Starte mit einzelnen Punkten und fasse diese dann sukzessive zu Clustern zusammen
  - Kombiniere immer die beiden Cluster, die den geringsten Abstand zueinander haben
- Divisiv: Starte mit einem Cluster und unterteile dann sukzessive
  - Teile immer in die beiden Teilcluster, die den größten Abstand zueinander haben



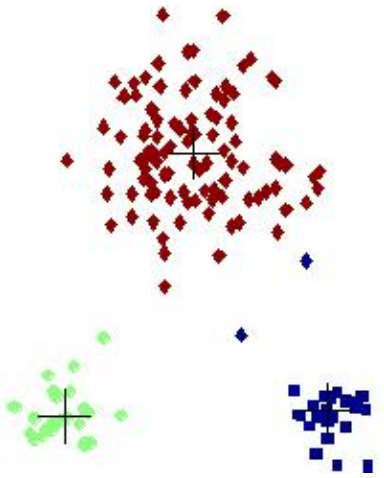
Dendrogramm

Einschub

K-Means Clustering

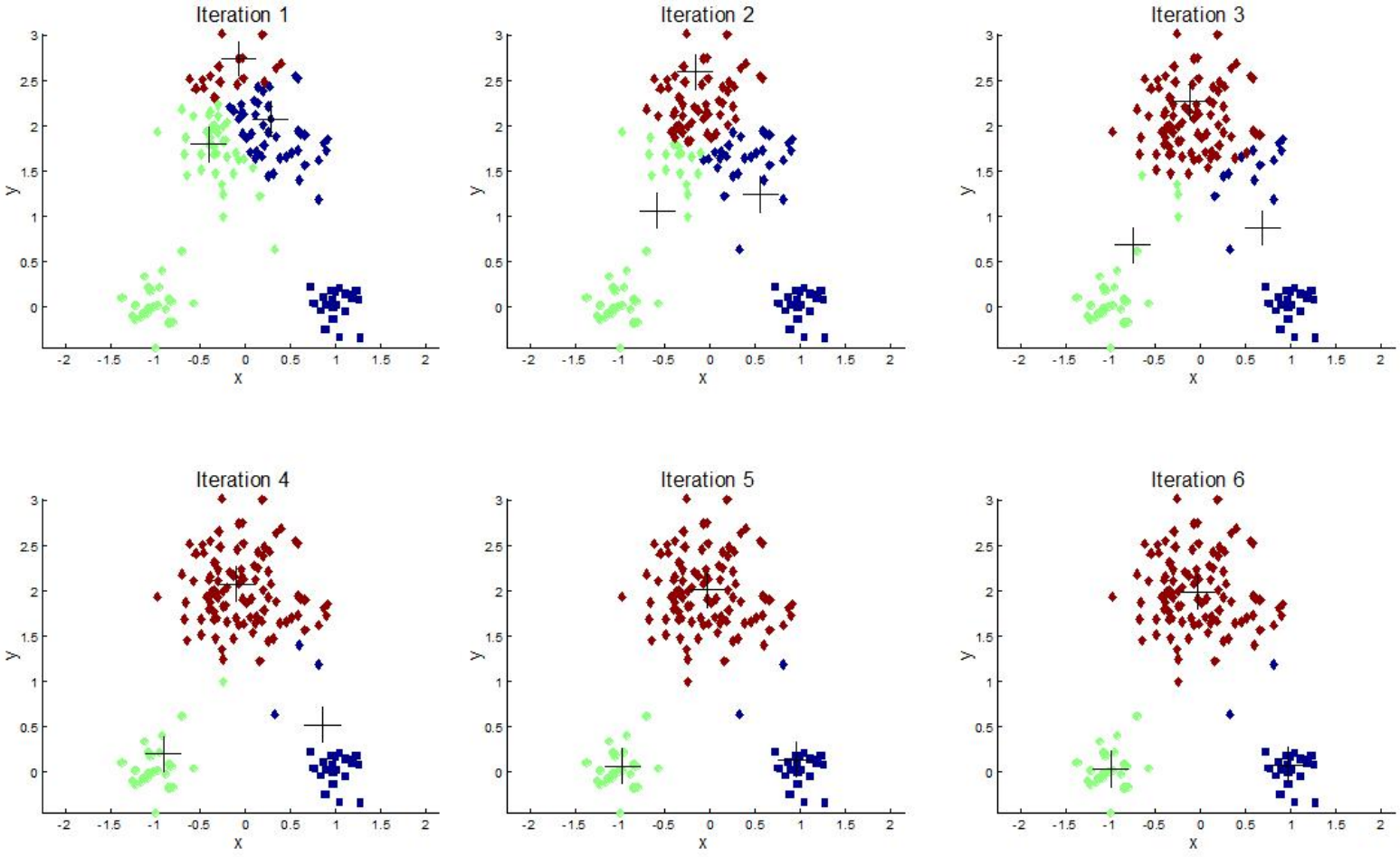
- Annahme: Wir wissen, dass es *k* Cluster in den Daten gibt!

- K-Means Algorithmus:
  1. Wähle *k* Datenpunkte als Clusterzentren aus
  2. Solange sich die Clusterzentren ändern
     - Ordne jeden Datenpunkt dem nächstgelegenen Clusterzentrum zu
     - Stelle sicher, dass jedem Clusterzentrum mindestens ein Punkt zugeordnet wurde
       - z.B. zufällige Zuordnung von weit entfernten Punkten
     - Berechne die Clusterzentren neu, indem sie durch den Schwerpunkt der zugeordneten Punkte ersetzt werden



Einschub

K-Means Clustering



Einschub

K-Means für 2D-„Segmentierung“

Farb-”Kompression”:

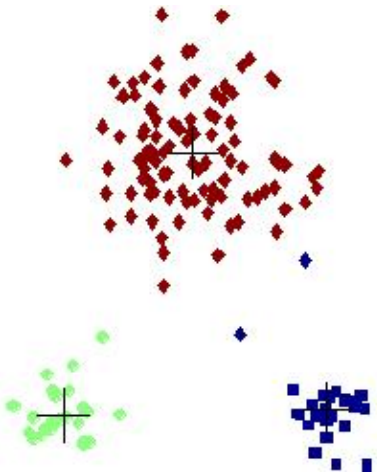
Segmentierung:

Nur Farbe

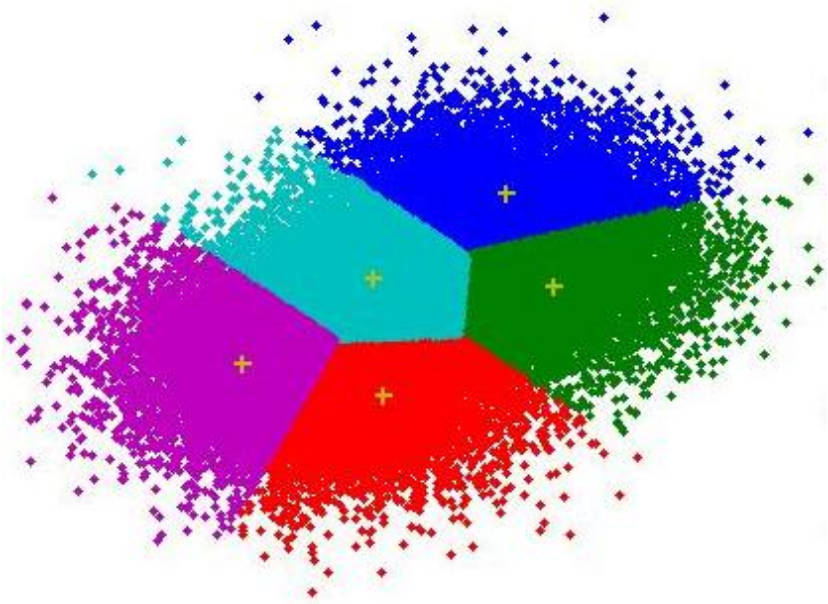
Farbe und Ort

Einschub

K-Means erzeugt soviele Cluster, wie vorgegeben!



Korrekte Segmentierung



Übersegmentierung

„Klassiker der 3D-Datenverarbeitung“

- Viele Algorithmen beinhalten die Suche des nächsten Punkts im m-dimensionalen Raum
  - Bisher: Region Growing, k-means, ICP, … u.v.m.
- Lineare Suche bei großen Punktemengen extrem aufwändig, linear in der Größe der Suchmenge
  - O(n*d): n ist die Anzahl der Punkte, d ist die Dimension des Raums
- Alternative: Suchbäume
  - Kd-Trees

2-dimensionale kd-Trees

- Eine Datenstruktur für räumliche Anfragen in $\mathbf{R^2}$
  - Nicht das theoretische Optimum
  - Wird aber in der Praxis sehr häufig verwendet

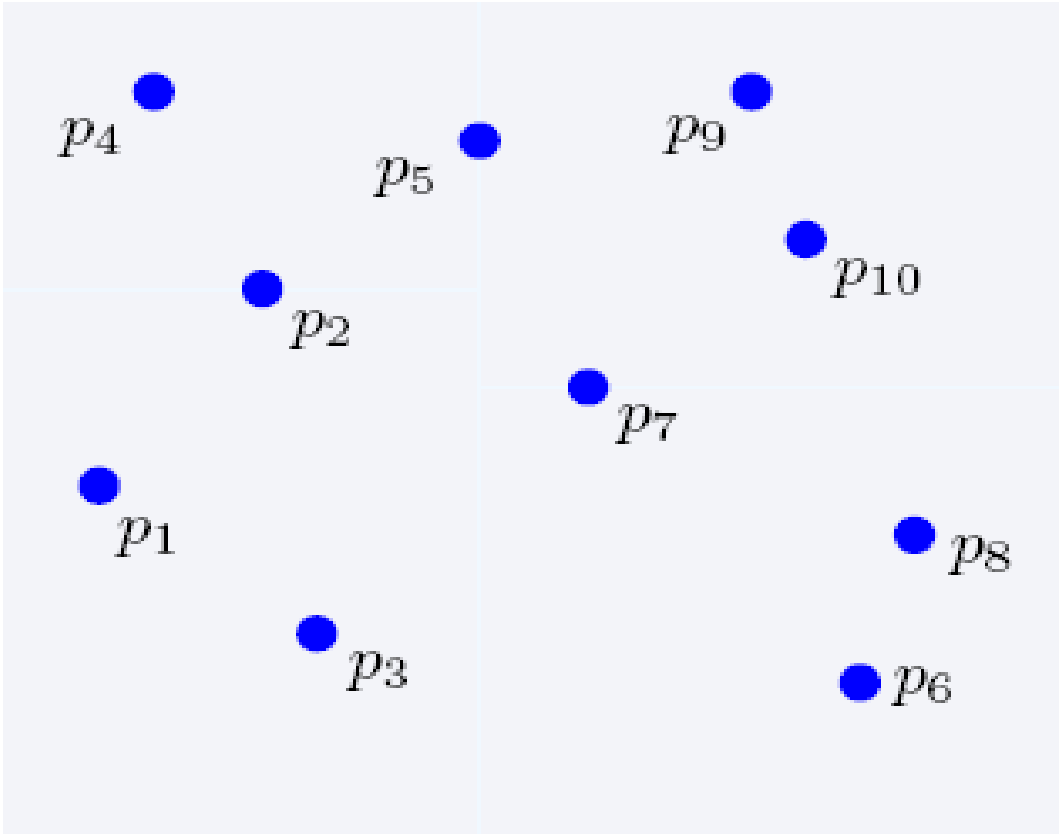
- Aufwand für Baumaufbau: **O(nlogn)**
- Speicheraufwand: **O(n)**
- Abfragezeit: $\mathbf{O(n^{1/2}+k)}$

2-dimensionale kd-Trees

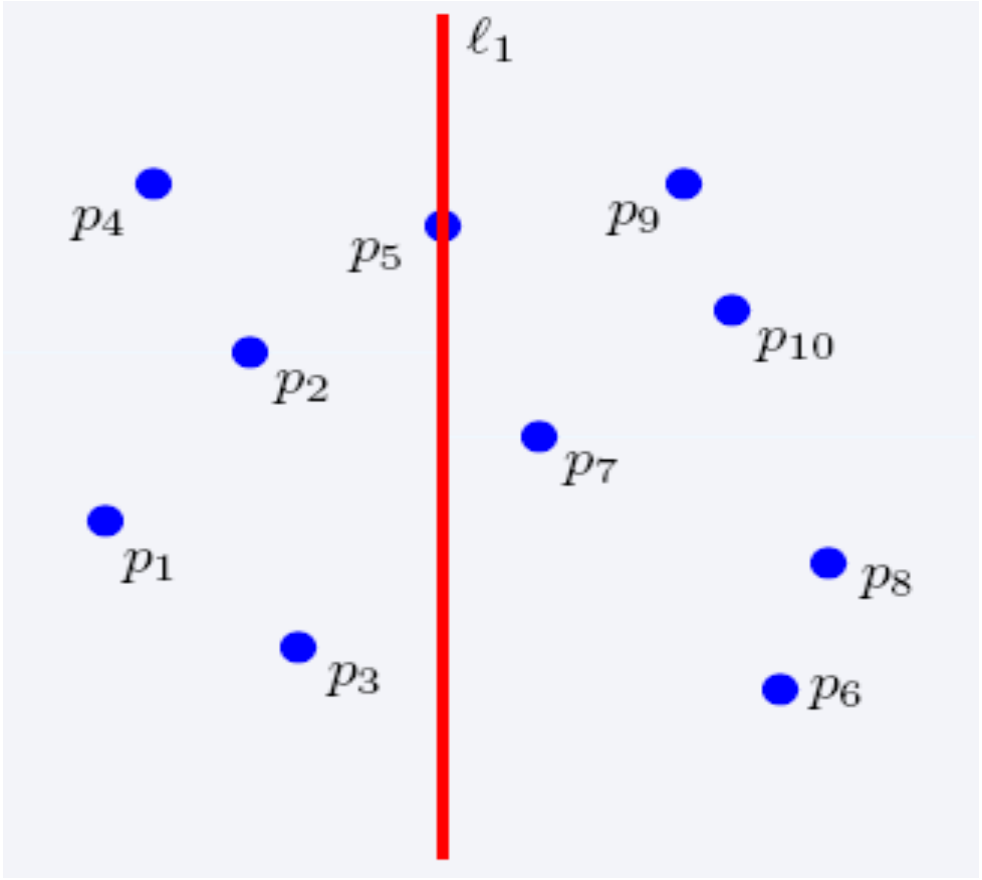
- Algorithmmus:
  - Wähle **x-** oder **y-** Koordinate (abwechselnd)
  - Wähle den Median der gewählten Koordinate; dies defniniert eine horizontale oder vertikale Gerade
  - Wiederhole dies auf beiden Seiten der Ebene rekursiv

- Daraus ergibt sich dann ein Binärbaum:
  - Größe **O(n)**
  - Tiefe **O(logn)**
  - Aufbau **O(nlogn)**

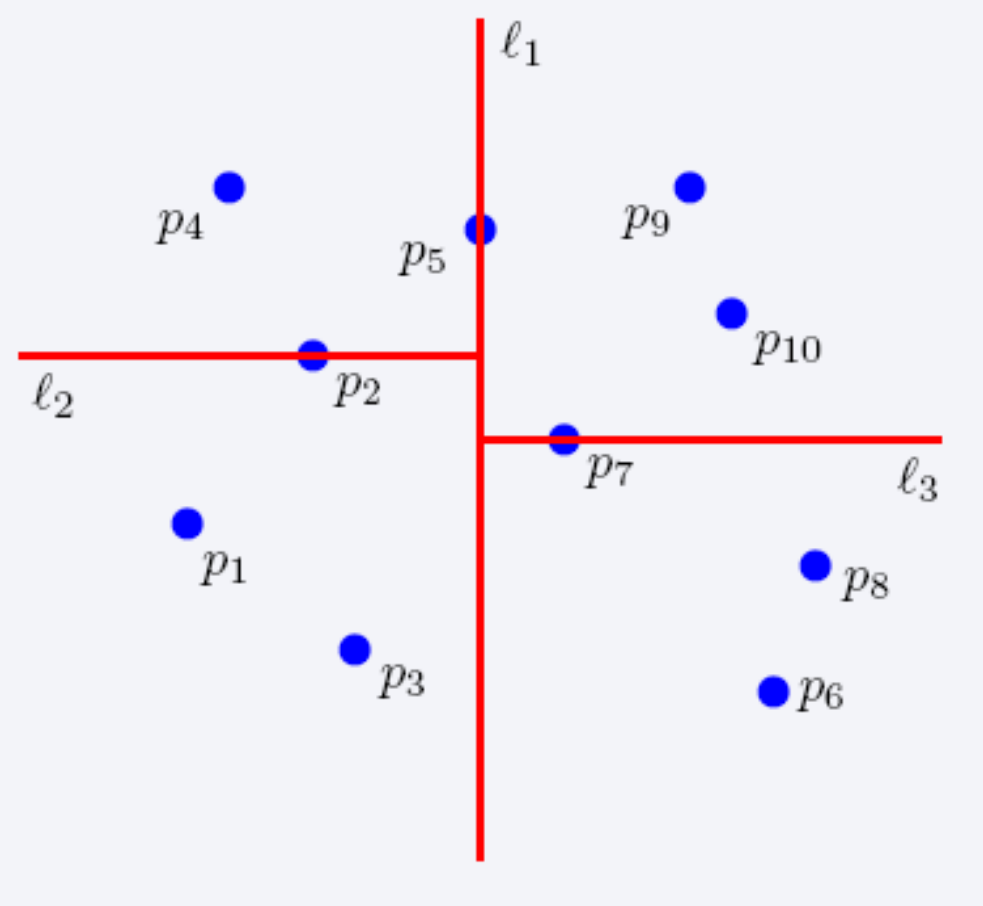
Aufbau von kd-Trees



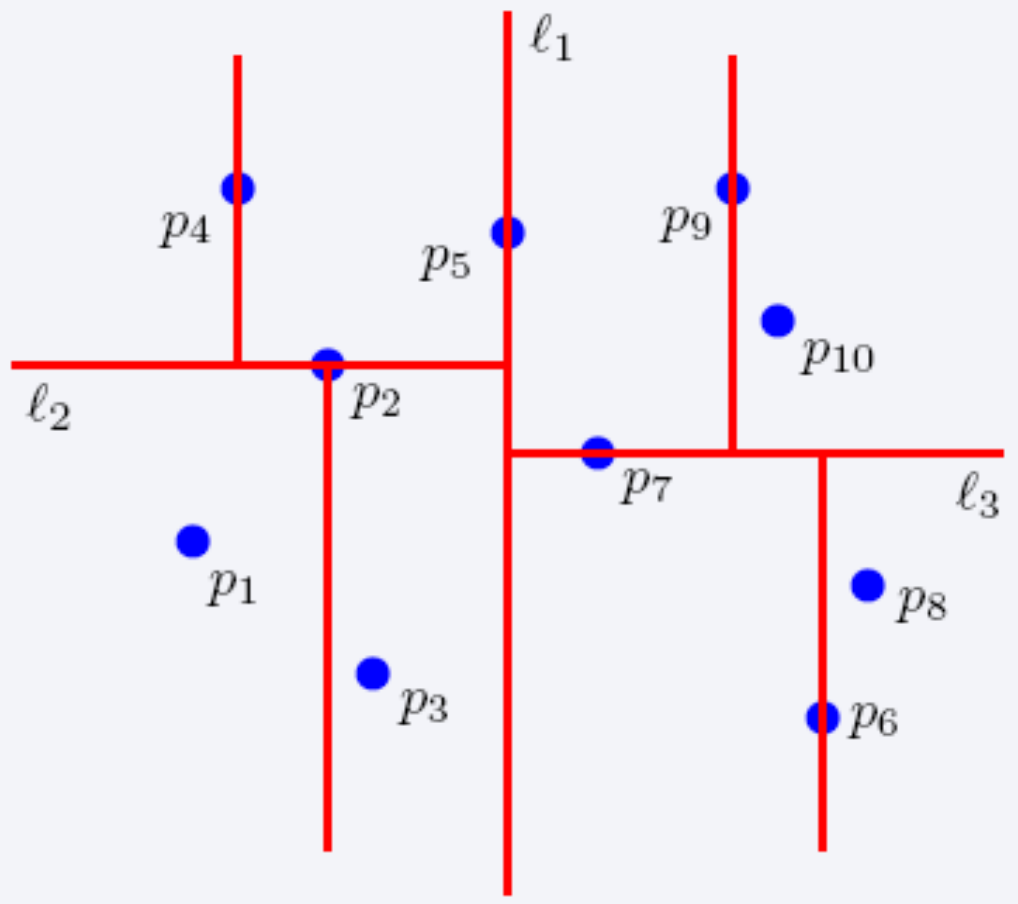
Aufbau von kd-Trees



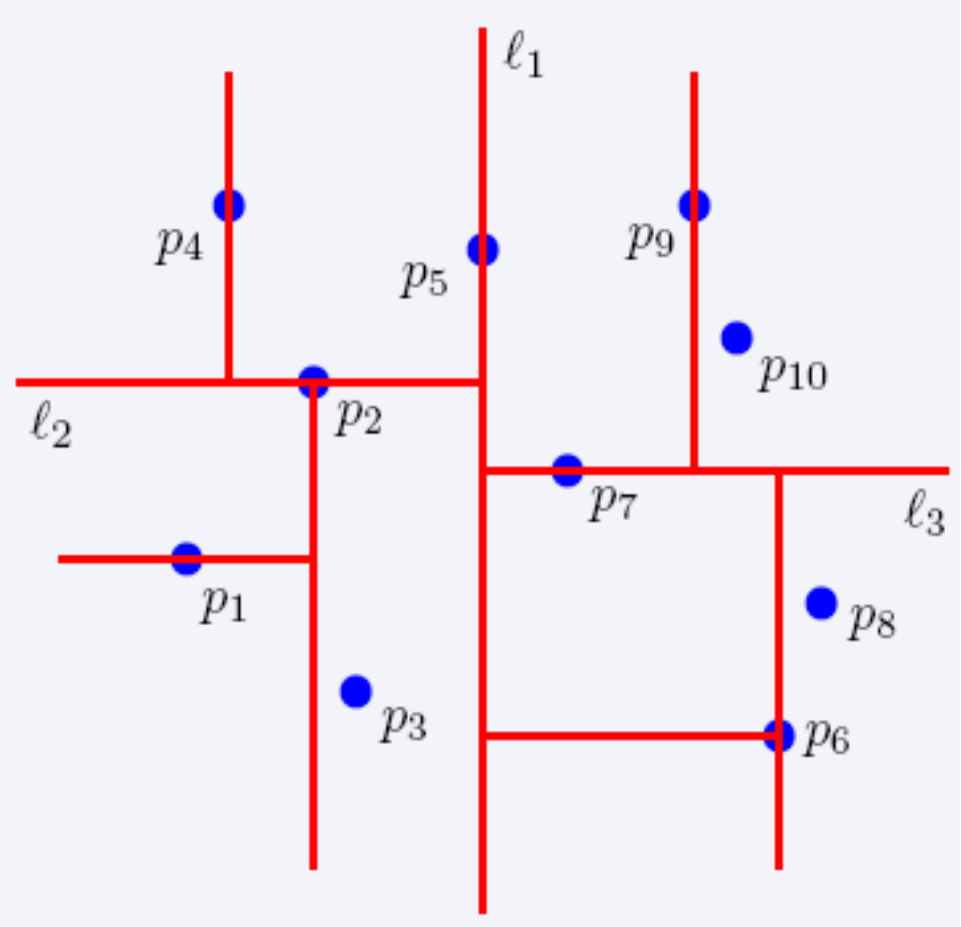
Aufbau von kd-Trees



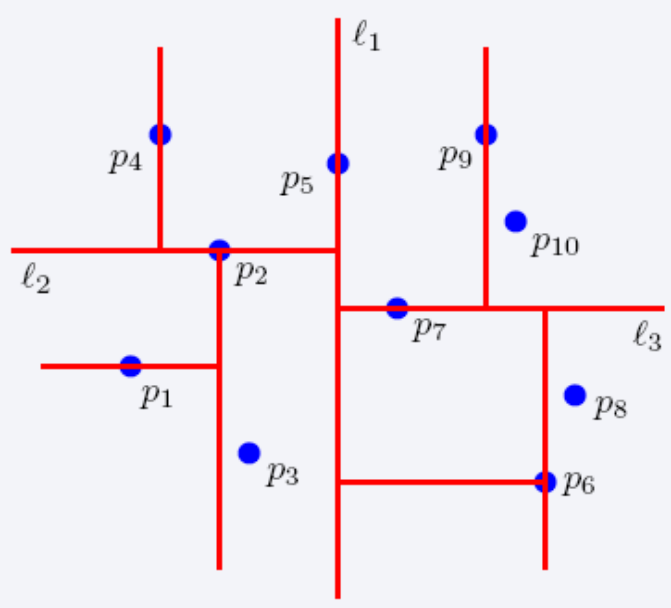
Aufbau von kd-Trees

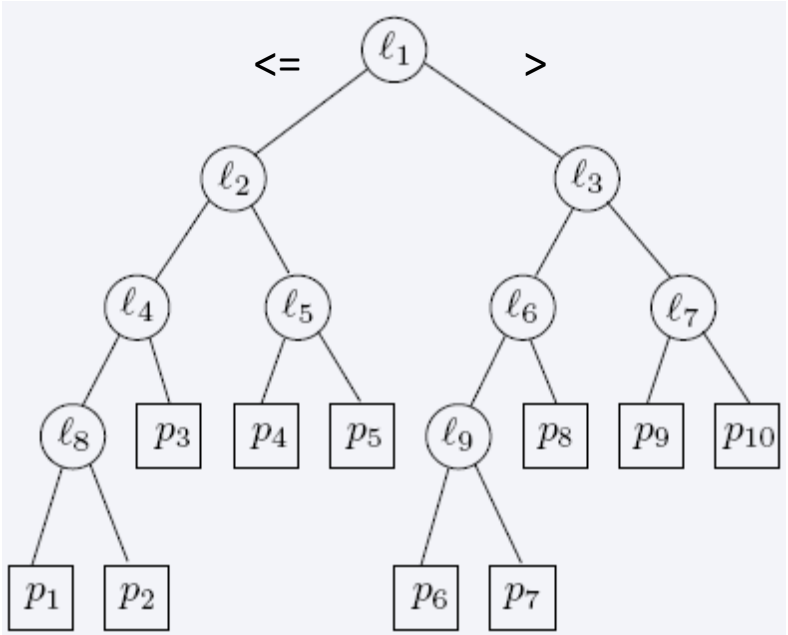


Aufbau von kd-Trees

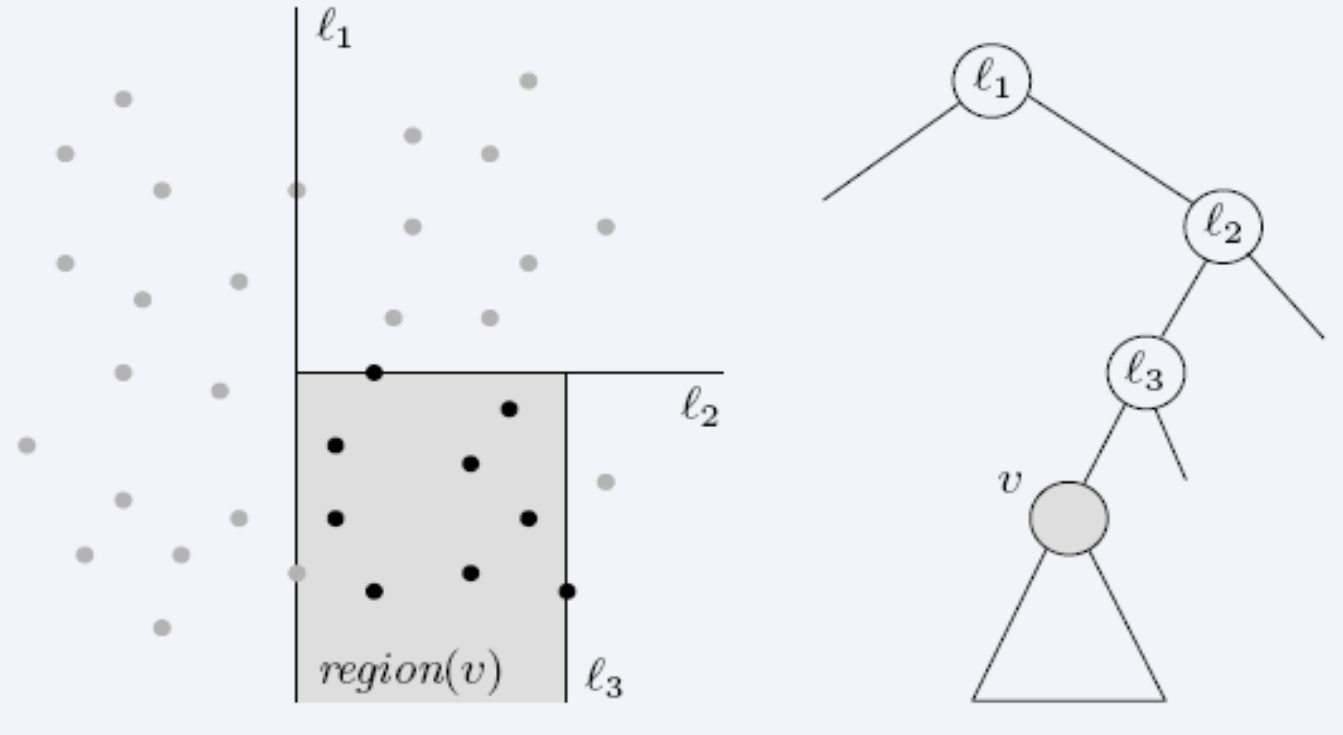


Der fertige kd-Tree





Zuständigkeitsregion des Knoten **v**



Region(v) : Der Teilbaum mit der Wurzel **v** speichert die schwarz eingetragenen Punkte

Suche in kd-Trees

- Bereichssuche in **2D**
  - Geben sei eine Menge von n Punkten. Erzeuge eine Datenstruktur, die für jedes Abfragerechteck **R** alle Punkte innerhalb von **R** liefert.

kd-tree: Bereichssuche

- Rekursives Vorgehen, startet bei **v = root**
- **Suche (v,R)**
  - Wenn **v** ein Blatt des Baumes ist, liefere alle Punkte innerhalb von **v**, falls es in **R** liegt
  - Anderenfalls, wenn **Region(v)** in **R** liegt, liefere alle Punkte im Teilbaum von **v**
  - Sonst:
    - Wenn die **Region(left(v))** die Abfrage **R** schneidet, dann **Suche (left(v),R)**
    - Wenn die **Region(right(v))** die Abfrage **R** schneidet, dann **Suche (right(v),R)**

d-dimensionale kd-Trees

- Eine Datenstruktur für räumliche Anfragen in $\mathbf{R^d}$

- Aufwand für Baumaufbau: **O(nlogn)**
- Speicherbedarf: **O(n)**
- Abfragezeit: $\mathbf{O(n^{1-1/d}+k)}$

Aufbau d-dimensionaler kd-Trees

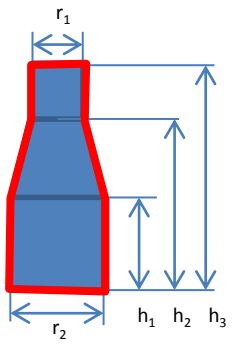
- Der Konstruktionsalgorithmus ist ähnlich wie in **2D**
- An der Wurzel teilen wir die Punkte in zwei gleich große Teilmengen, getrennt durch eine Hyperebene senkrecht zur x_1-Achse
- Die etntsandenen Teilmengen teilen wir dann entsprechend der zweiten Koordinatenrichtung: snkrecht zur x_2-Achse
- …
- Bei Baumtiefe **d**, beginnen wir wieder mit der ersten Koordinate
- Die Rekursion stops läuft, bis nur noch ein Punkt übrig ist, dieser wird als “Blatt” des Baums abgelegt.

kd-Trees

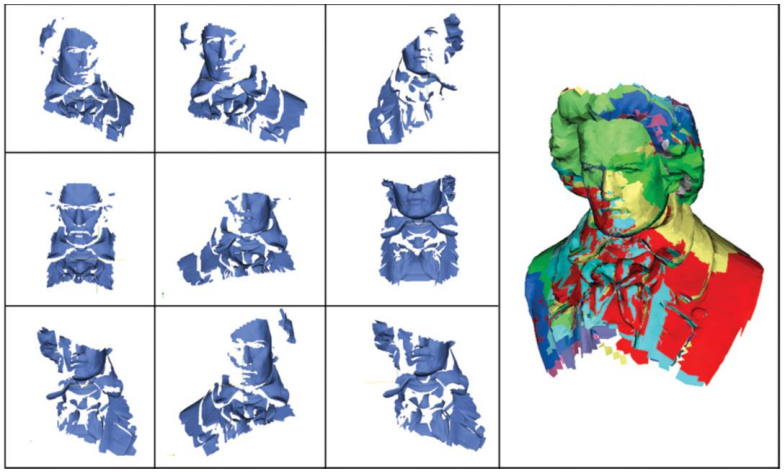
- Kd-Trees funktionieren gut bei wenig bis mittel-dimensionalen Räumen
- In der Praxis sehr verbreitet
- Zahllose Varianten: Vor allem hinsichtlich Baumaufbau (Heuristiken…)

- Weit verbreitet zur Beschleunigung der Nächster-Nachbar-Suche!

„Klassiker der 3D-Datenverarbeitung“



- Letze Woche: Parametrisches Modell der Flasche
- Alternative: Modellierung durch Registrierung von 3D Punktewolken
- „Iterative Closest Points“-Algorithmus



Paul J. Besl und Neil D. McKay, *A Method for Registration of 3-D Shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (2), Feb. 1992

ICP-Algorithmus

- Ausgangspunkt: Zwei Punktemengen

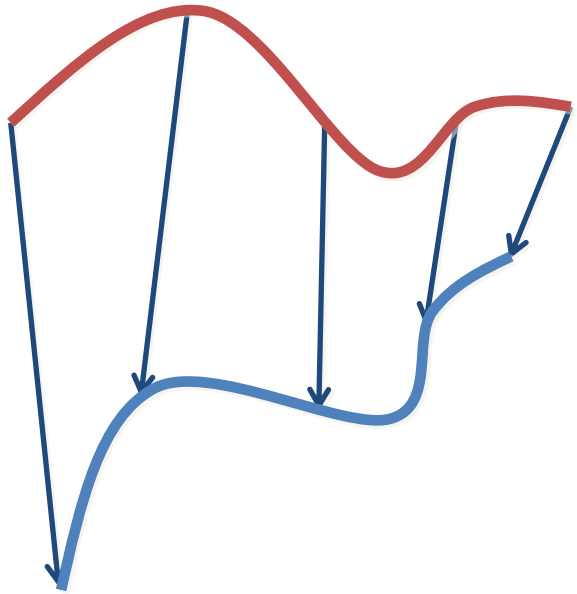
$$Y := \{y_1, ..., y_n\} \quad X = \{x_1, ..., x_m\}$$

Messung Modell

- Abstandsmaß: Punkt zu Modell

$$d(y_i, X) = \min_{x_j \in X} ||x_j - y_i||$$

- Transformation auftrennen nach Translation und Rotation



Optimale Rotation (bei bekannten Korrespondenzen)

- Transformation der Modellpunkte auf die gemessenen Punkte mit der Rotationsmatrix *R*, dem Translationsvektor *t*, der Skalierung *c* und dem unvermeidlichen Rauschen s_i des jeweiligen Punktes

$$y_i = cRx_i + t + s_i, \quad i = 1...n$$

- Fehlerfunktion

$$E^2 = \frac{1}{n}\sum_{i=1}^{n} \|y_i - (cRx_i + t)\|^2.$$

- Verschiebung beider Punktemengen in ihren jeweiligen Schwerpunkt

$$\mu_X := \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \mu_Y := \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$q_{Xi} := x_i - \mu_X, \quad q_{Yi} := y_i - \mu_Y$$

- Gesucht: Optimale Transformation (im Sinne des minimalen quadratischen Fehlers E^2)

$$\hat{R}, \hat{t}, \hat{c} = \underset{R,c,t}{\operatorname{argmin}} E^2$$

Optimale Rotation (bei bekannten Korrespondenzen)

- Verschiebung beider Punktemengen in ihren jeweiligen Schwerpunkt

$$q_{Xi} := x_i - \mu_X, \quad q_{Yi} := y_i - \mu_Y$$

- Neue Fehlerfunktion (ohne Translation)

$$E^2 = \sum_{i=1}^{n} \|q_{Yi} - \hat{c}\hat{R}q_{Xi}\|^2$$

- Kreuzkovarianmatrix der Punkte

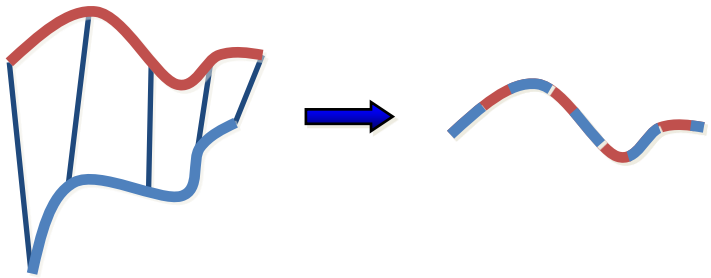
$$D = \frac{1}{n}\sum_{i=1}^{n} q_{Yi} * q_{Xi}^T$$

- Varianz der Modellpunkte

$$\sigma_X^2 = \frac{1}{n}\sum_{i=1}^{n} \|q_{Xi}\|^2$$

- Lösung des Minimierungsproblems über Singulärwertzerlegung von *D*

$$D = USV^T$$ *S*: Diagonalmatrix



Nächste Folie!

$$\hat{R}, \hat{t}, \hat{c} = \underset{R,c,t}{\operatorname{argmin}} E^2$$

Optimale Rotation (bei bekannten Korrespondenzen)

- Lösung des Minimierungsproblems über Singulärwertzerlegung von *D*
  - Finde *U*, *S* und *V*, sodass $D = USV^T$

Singular Value Decomposition (SVD)

S: Diagonalmatrix

$$\hat{R} = UK_1V^T, \quad K_1 = \begin{cases} I, & \text{falls } det(D) \geq 0, \\ diag(1, ..., 1, -1), & \text{falls } det(D) < 0. \end{cases}$$

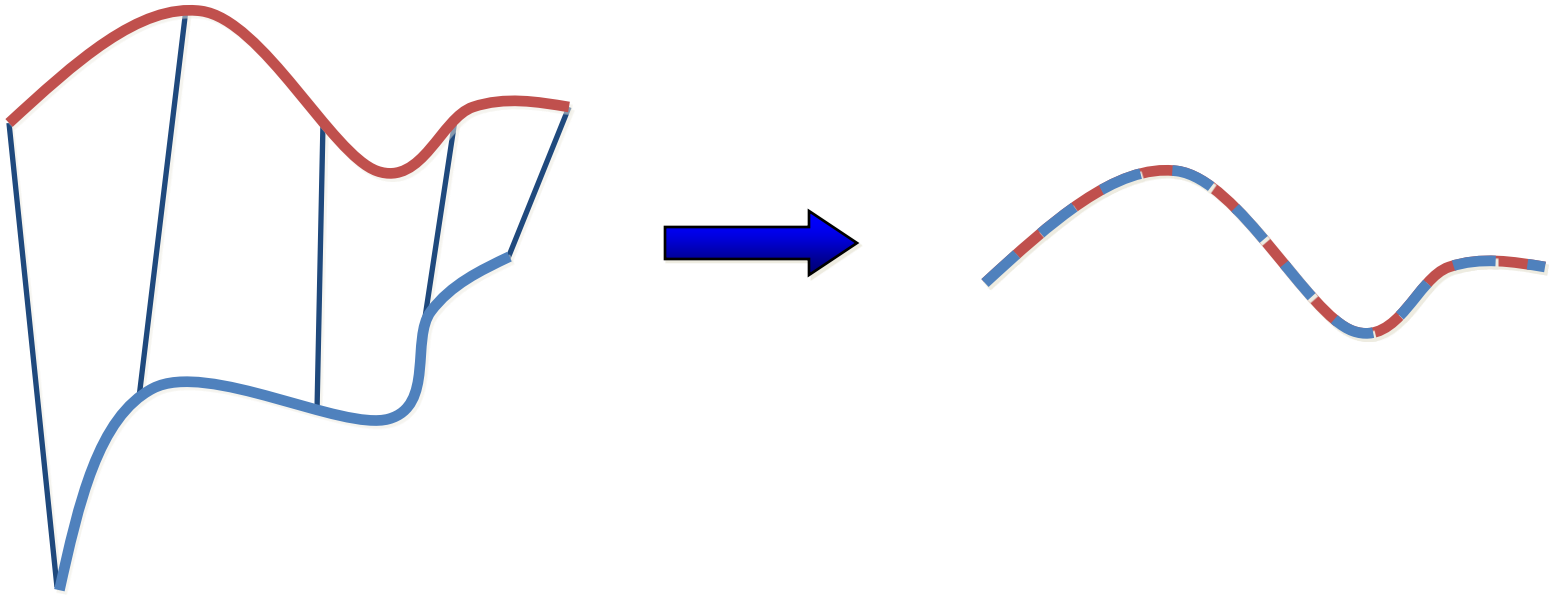
$$\hat{c} = \frac{1}{\sigma_X^2} spur(SK_2), \quad K_2 = \begin{cases} I, & \text{falls } det(U)det(V) = 1, \\ diag(1, ..., 1, -1), & \text{falls } det(U)det(V) = -1. \end{cases}$$

$$\hat{t} = \mu_Y - \hat{c}\hat{R}\mu_X$$

Umeyama, Shinji. "Least-squares estimation of transformation parameters between two point patterns." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 13.4 (1991): 376-380.

ICP-Algorithmus

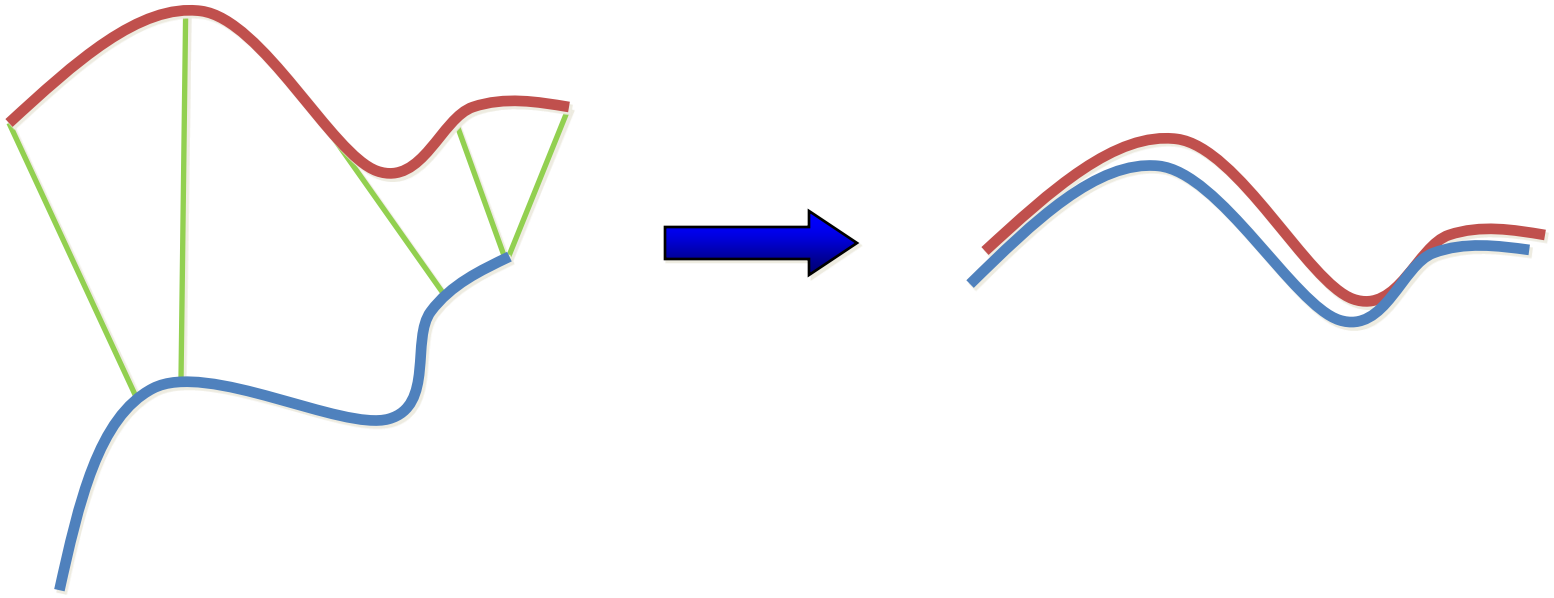
Wenn die Punktkorrespondenzen bekannt sind, kann man die optimale Transformation berechnen (siehe vorhergehende Folien)



ICP-Algorithmus

- Wie finden wir die richtigen Korrespondencen:
  - User input? Merkmalsdetektion? Signaturen?

- Alternative: Wir nehmen einfach an, dass immer die nächsten Punkte zusammen gehören!

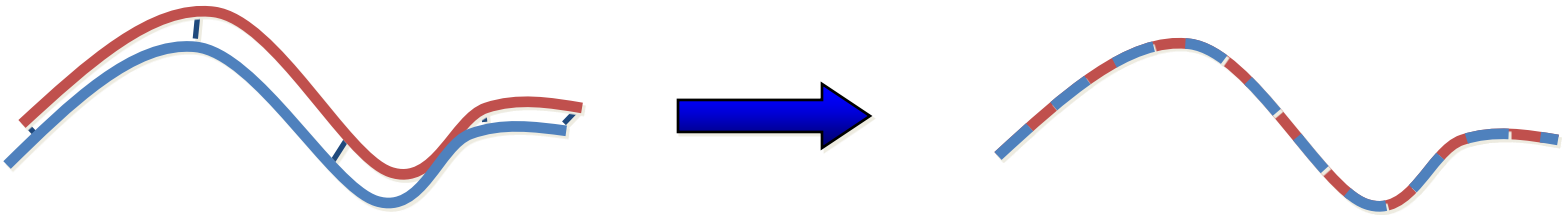


ICP-Algorithmus

- … und wiederholen das solange, bis der Restfehler E^2 unter eine geeignet zu wählende Schwelle fällt
  - Iterative Closest Points (ICP)

Paul J. Besl und Neil D. McKay, *A Method for Registration of 3-D Shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (2), Feb. 1992

- Konvergiert aus „ausreichend guter“ Startposition
  - Problem: Lokale Minima!

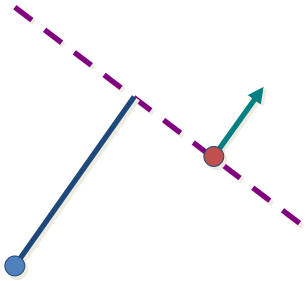


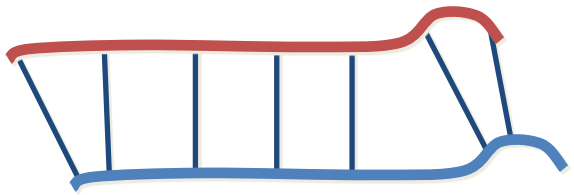
ICP-Algorithmus

1. Verschiebe beide Punktemengen in ihren Schwerpunkt
2. Wähle z.B. 1000 zufällige Punkte aus dem Modell
3. Ordne jeden dieser Punkte seinem nächsten Nachbarn aus der Messung zu
   - Optional, aber sinnvoll: Ignoriere Paare, deren Abstand > k-mal der Median aller Abstände ist
4. Berechne die optimale Transformation $\hat{R}, \hat{t}, \hat{c} = \underset{R,c,t}{\operatorname{argmin}} E^2$
5. Gehe zu 2. solange der Restfehler E^2 über einer Schwelle t liegt oder eine Anzahl N_{max} von Iterationen nicht überschritten wurde

ICP-Algorithmus

- Wie immer bei derartigen Basisalgorithmen, gibt es zahllose Varianten
  - Hier: Punktmenge zu Punktmenge
  - Andere: Punktmenge zu parametrischem Modell, es muß möglich sein Abstände der Punkte zum Modell zu berechnen
  - Beispiel: Punkt zu Ebene, seitliche Verschiebungen erhöhen den Fehler nicht (Vorteil!), erfordert lokale Normalenschätzung



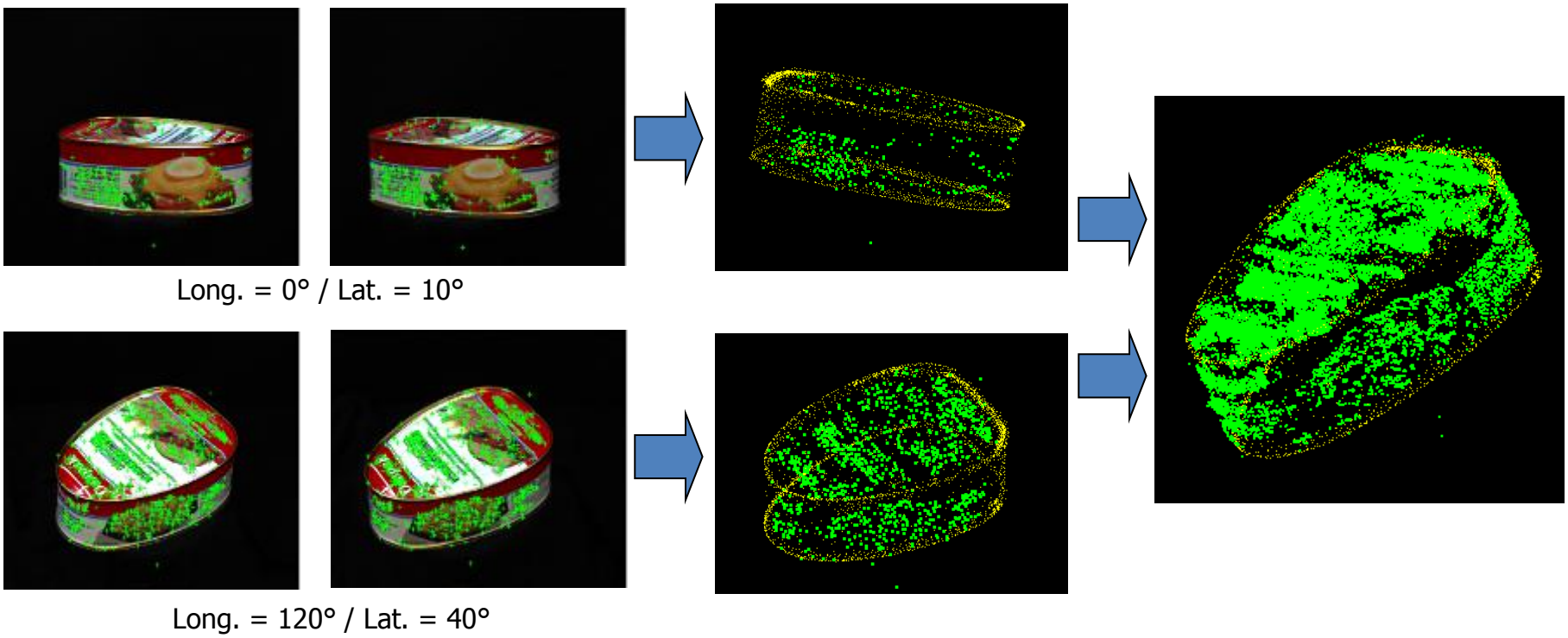


ICP-Anwendungsbeispiel

- Modellierung von 3D Objekten

ICP-Anwendungsbeispiel

- Modellierung von 3D Objekten
  - Registrierung von Teilpunktewolken aus unterschiedlichen Ansichten



Diese Woche

- Ein paar Algorithmenklassiker für den Umgang mit Punktewolken
  - Clusterbildung mit k-Means und Agglomeration
  - Registrierung mit ICP
  - Nachbarschaftsuche mit dem kd-Tree

- Kamerageometrie