Vorlesung: Angewandte Sensorik

Prof. J. Zhang
zhang@informatik.uni-hamburg.de

Universität Hamburg
Fachbereich Informatik
AB Technische Aspekte Multimodaler Systeme

09. Dezember 2005

Verarbeitung von Abstandsmessungen

Für Messungen mit Lasermesssystemen gibt es Verfahren zur

- Extraktion von Liniensegmenten,
- Extraktion von Ecken,
- Klassifikation von Scanpunkten,
- Filterung von Scans:
  - Glätten,
  - Datenreduktion.

Inhaltsverzeichnis

Scan (1)

Definition:

Ein Scan ist eine Menge von Messwerten

$$\left\{m_i = (\alpha_i, r_i)^T | i = 0 \ldots n-1\right\},$$

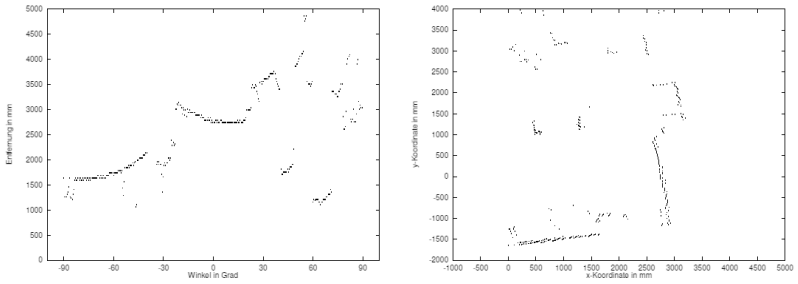
welche in Polarkoordinaten $(\alpha_i, r_i)^T$ angegeben sind.

Scan (2)

Ein Scanpunkt $m_i = (\alpha_i, r_i)^T$ kann für eine gegebene Aufnahmeposition $l = (x, y, \theta)^T$ in absolute Koordinaten umgerechnet werden.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} r_i \cos\alpha_i \\ r_i \sin\alpha_i \end{bmatrix}$$

Scan (3)



Filtern von Scandaten

- Problem von Scandaten: große Datenmenge, ungewünschte Scanpunkte.
- Daher oftmals vorab Filterung von Scandaten.
- gängige Filter:
  - Medianfilter
  - Reduktionsfilter
  - Winkelreduktionsfilter
  - Linienfilter

Medianfilter (1)

- Der *Medianfilter* erkennt Ausreißer und ersetzt diese durch eine geeignete Messung.
- Um jeden Scanpunkt wird ein Fenster gelegt, das die Messungen vor und nach dem Punkt enthält.
- Der Scanpunkt wird ersetzt durch einen Punkt, der denselben Aufnahmewinkel, aber den Median der Entfernungsmessungen des betrachteten Fensters als Entfernung hat.
- MEDIAN-NUM-POINTS bestimmt die Fenstergröße.
- Ein großer Wert bedeutet eine starke Glättung.
- Nachteil des Medianfilters: Ecken werden abgerundet.

Medianfilter (2)

Algorithmus: Medianfilter
Eingabe: Scan s
Ausgabe: Scan s'

for $i := 0$ **to** numpoints(s)-1 **do**
  $p :=$ n-th-scanpoint(s,i)
  for $j := 0$ **to** MEDIAN-NUM-POINTS-1 **do**
    $k := (i + j$ - MEDIAN-NUM-POINTS/2) *mod* numpoints(s)
    $p_k :=$ n-th-scanpoint(s,k)
    $d(j) :=$ distance-value(p_k)
  endfor
  $d_{median} :=$ median(d)
  n-th-scanpoint(s',i) := (angle-value(p),d_{median})
endfor
return s'

Medianfilter (3)

Reduktionsfilter (1)

- Der *Reduktionsfilter* fasst Punktwolken zu einem Punkt zusammen.
- Eine Punktwolke wird durch einen Radius r angegeben.
- Der erste Punkt (Ausgangspunkt) eines Scans gehört zu einer Wolke.
- Alle folgenden Punkte mit Abstand $d < 2 \cdot r$ werden zur Wolke hinzugefügt.
- Beim ersten Punkt mit größerem Abstand wird eine neue Wolke angefangen.
- Jede Wolke wird durch den Schwerpunkt der ihr zugeordneten Punkte ersetzt.

Reduktionsfilter (2)

Algorithmus: Reduktionsfilter
Eingabe: Scan s, Radius r
Ausgabe: Scan s'

$j := 0$
$p_0 :=$ n-th-scanpoint(s,0)
$p_{sum} := p_0$
$n := 0$
for $i := 1$ **to** numpoints(s)-1 **do**
  $p :=$ n-th-scanpoint(s,i)
  if distance(p_0,p) $< 2r$ **then**
    $p_{sum} := p_{sum} + p$
    $n := n + 1$
  else
    n-th-scanpoint(s',j) := p_{sum}/n
    $j := j + 1$
    $P_0 := p$

$p_{sum} := p_0$
$n := 1$
endif
endfor
n-th-scanpoint(s',j) := p_{sum}/n
$j := j + 1$
numpoints(s') := j
return s'

Reduktionsfilter (3)

Reduktionsfilter (4)

- Der Algorithmus des Reduktionsfilters hat eine Zeitkomplexität von $O(n)$, wenn n die Anzahl der Punkte ist.
- Vorteile des Reduktionsfilters:
  - Reduzierung der Anzahl der Scanpunkte ohne Verlust wesentlicher Informationen.
  - Dies führt zu kürzeren Laufzeiten bei der Nachbearbeitung eines Scans.
  - Es ergibt sich eine bessere Gleichverteilung der Punkte.
- Nachteile des Reduktionsfilters:
  - Extraktion von Merkmalen nicht mehr so einfach.
  - Möglicherweise zu wenig Punkte für Merkmal.
  - Daher besser: Merkmalsextraktion vor Reduktionsfilter.

Winkelreduktionsfilter (1)

- Der *Winkelreduktionsfehler* ist dem Reduktionsfilter sehr ähnlich.
- Scanpunkte mit ähnlichem Aufnahmewinkel werden gruppiert und durch den Punkt ersetzt, dessen Entfernung gleich dem Median der Entfernungswerte ist.
- Die Funktion $median - dist(q, n)$ liefert im folgenden Algorithmus diesen Punkt.
- Die Zeitkomplexität ist $O(n)$, wenn n die Anzahl der Scanpunkte ist.
- Der Winkelreduktionsfilter wird eingesetzt um Scans mit hoher Winkelauflösung gleichmäßig zu reduzieren.
- Dies wird z.B. bei der Verschmelzung mehrerer Scans zu einem Scan eingesetzt.

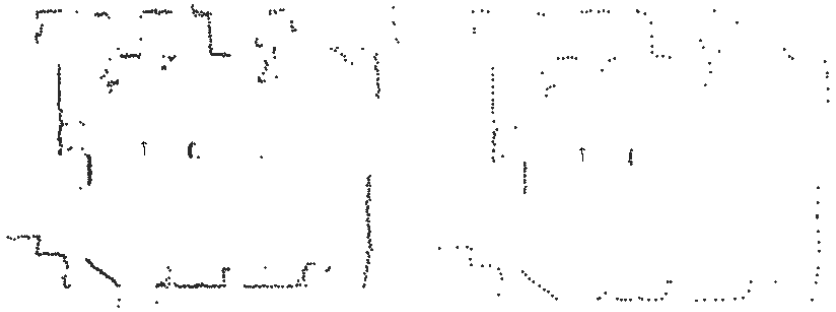
Winkelreduktionsfilter (2)

Algorithmus: Winkelreduktionsfilter
Eingabe: Scan s, Winkel α
Ausgabe: Scan s'

$j := 0$
$q(0) :=$ n-th-scanpoint(s,0)
$n := 1$
for $i := 1$ **to** numpoints(s)-1 **do**
  $p :=$ n-th-scanpoint(s,i)
  if *abs*(angle-value(p) - angle-value($q(0)$)) $< \alpha$ **then**
    $q(n) := p$
    $n := n + 1$
  else
    n-th-scanpoint(s',j) := median-dist(q,n)
    $j := j + 1$
    $q(0) := p$
    $n := 1$

  endif
endfor
n-th-scanpoint(s',j) := median-dist(q,n)
$j := j + 1$
numpoints(s') := j
return s'

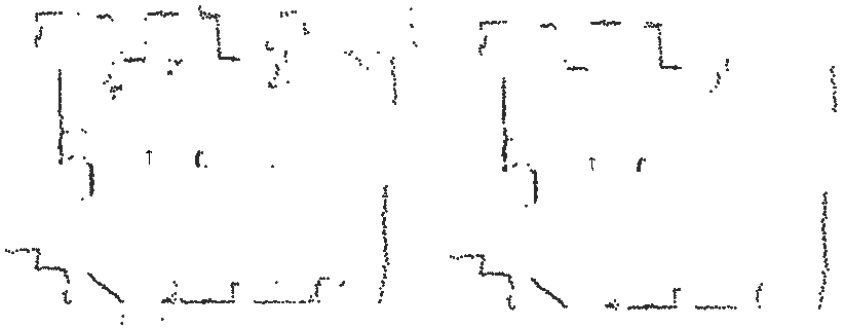
Winkelreduktionsfilter (3)



Linienfilter (1)

- Der *Linienfilter* nutzt das später vorgestellte Verfahren zur Linienextraktion.
- Die Scanpunkte, die keinem Liniensegment zugeordnet wurden, werden entfernt.
- Die Zeitkomplexität ist gleich der Komplexität der Linienextraktion ($O(n \log n)$ im mittleren Fall).
- Der Filter wird angewendet, wenn anschließend angewandte Algorithmen polygonale Umgebungen erfordern.

Linienfilter (2)



Merkmalsextraktion

- Keine Verarbeitung von kompletten Scans sondern Merkmalsextraktion.
- Häufige Merkmale: Linien, Ecken
- MAX-DISTANCE im folgenden Algorithmus ist der maximal zulässige Abstand zweier aufeinander folgender Punkte für die Gruppierung.

Linien (1)

Algorithmus: Linienextraktion
Eingabe: Scan s
Ausgabe: Menge von Linien l

$l :=$ empty
$start := 0$
for i:=1 **to** numpoints(s)-1 **do**
  $p_1 :=$ n-th-scanpoint(s,i-1)
  $p_2 :=$ n-th-scanpoint(s,i)
  if distance(p_1,p_2) $>$ MAX-DISTANCE **then**
    $l := l \cup$ split(s,$start$,i-1)
    $start := i$
  endif
endfor
$l := l \cup$ split(s,$start$,numpoints(s)-1)
return l

Linien (2)

Algorithmus: split(s,$start$,end)
Eingabe: Gruppe von Scanpunkten, festgelegt durch s, $start$ und end
Ausgabe: Menge von Linien l

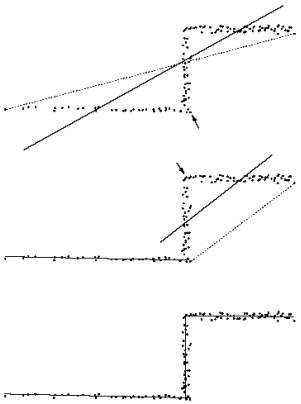
$l :=$ empty
$line :=$ make-line(s,$start$,end)
if numpoints($line$) $\geq$ MIN-POINTS-ON-LINE **then**
  if $\sigma(line) <$ MAX-SIGMA **then**
    $l := l \cup \{line\}$
  else
    $p_{start} :=$ n-th-scanpoint(s,$start$)
    $p_{end} :=$ n-th-scanpoint(s,end)
    $i_{split} := start$
    $d := 0$
    for $i := start$+1 **to** end-1 **do**
      $p :=$ n-th-scanpoint(s,i)

```
            if distance-to-line(p,p_start,p_end) > d then
                i_split := i
                d := distance-to-line(p,p_start,p_end)
            endif
        endfor
        l := l ∪ split(s,start,i_split)
        l := l ∪ split(s,i_split,end)
    endif
endif
return l
```


Linien (3)

- Die $split$-Funktion ist rekursiv.
- Zuerst wird eine Ausgleichsgerade durch die Punkte gelegt.
- Ist die Abweichung $\sigma(line)$ zu groß wird die Punktmenge aufgeteilt und für die neuen Mengen die Funktion $split$ aufgerufen.
- Der Punkt an dem aufgeteilt wird, ist der Punkt mit dem größten Abstand zur Geraden durch $start$ und end.
- MIN-POINTS-ON-LINE und MAX-SIGMA bestimmen die Anzahl und Qualität der Linien.
- Die Linienextraktion ist ein typischer *divide and conquer* Algorithmus.
- Zeitkomplexität ähnlich *Quicksort*: $O(n^2)$ im schlechtesten, $O(n \log n)$ im mittleren Fall (n: Anzahl der Scanpunkte).

Linien (4)



Linien (5)

Ecken (1)

- Ähnlich wie Linienalgoritmus.
- Aufeinander folgende Linien werden miteinander geschnitten.
- Nur Ersetzen der $split$-Funktion notwendig.
- Gleiche Zeitkomplexität.

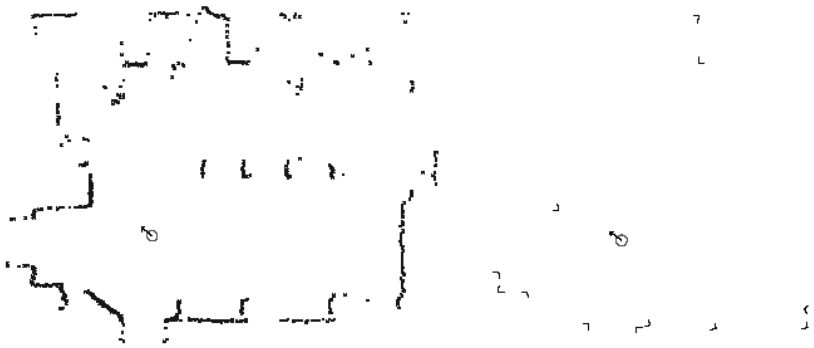
Ecken (2)

Algorithmus: split(s,$start$,end)
Eingabe: Gruppe von Scanpunkten, festgelegt durch s, $start$ und end
Ausgabe: Menge von Ecken e

e := empty
if (end -$start$) $\geq 2 \cdot$ MIN-POINTS-CORNER **then**
  p_{start} := n-th-scanpoint(s,$start$)
  p_{end} := n-th-scanpoint(s,end)
  i_{split} := $start$
  d := 0
  for i := $start$+1 **to** end-1 **do**
    p := n-th-scanpoint(s,i)
    if distance-to-line(p,p_{start},p_{end}) $> d$ **then**
      i_{split} := i
      d := distance-to-line(p,p_{start},p_{end})
    endif
  endfor

if (i_{split} - $start$) $\geq$ MIN-POINTS-CORNER **and**
(end - i_{split}) $\geq$ MIN-POINTS-CORNER **then**
  $line_1$:= make-line(s,i_{split} - MIN-CORNER-POINTS,i_{split})
  $line_2$:= make-line(s,i_{split}, i_{split} + MIN-CORNER-POINTS)
  if $\sigma(line_1) <$ MAX-SIGMA and $\sigma(line_2) <$ MAX-SIGMA **then**
    $e := e \cup$ {make-corner($line_1$,$line_2$)}
  endif **endif**
$e := e \cup$ split(s,$start$,i_{split})
$e := e \cup$ split(s,i_{split},end)
endif
return e

Ecken (3)



Scan-Matching

- In der mobilen Robotik werden Laserscans häufig eingesetzt, um die Position des Roboters in einer Karte zu finden.
- Dazu wird der Scan mit Hilfe der Linienextraktion in eine Menge von Linien umgewandelt.
- Die gemessene Anordnung der Linien wird in einer Karte durch Überdeckung gesucht.
- Dieses Verfahren wird *Scan-Matching* genannt.

Verfahren von Cox (1)

- Einer der ersten Vorschläge zum Überdecken von Scandaten und einem *a priori* Linienmodell stammt von Cox (1990, 1991).
- Hierbei wird jedem Scanpunkt eine Linie des *a priori* Modells zugeordnet.
- Aus der Zuordnung lässt sich die Rotation und Translation gegenüber dem Linienmodell bestimmen.
- Das Verfahren benötigt eine ungefähre Anfangsschätzung der Aufnahmeposition z.B. anhand der Odometriedaten.

Verfahren von Cox (2)

1. Setze $(\hat{x}, \hat{y}, \hat{\theta})^T = (s_x, s_y, s_\theta)^T$, wobei $(s_x, s_y, s_\theta)^T$ die initiale Positionsschätzung der Scanaufnahme anhand der Odometrie ist.
2. Verschiebe und drehe Scan auf Position $(\hat{x}, \hat{y}, \hat{\theta})^T$.
3. Bestimme für jeden Scanpunkt die Modellinie, die dem Punkt am nächsten liegt. Diese Modellinie wird im folgenden Ziellinie genannt.
4. Berechne die Transformation $\hat{b} = (\delta x, \delta y, \delta\theta)^T$, welche die Summe der Abstandsquadrate zwischen Scanpunkten und jeweiliger Ziellinie minimiert.
5. Setze $(\hat{x}, \hat{y}, \hat{\theta})^T = (\hat{x}, \hat{y}, \hat{\theta})^T + (\delta x, \delta y, \delta\theta)^T$.
6. Wiederhole Schritte 2-5 bis das Verfahren konvergiert. Das Ergebnis der Überdeckung ist $(\hat{x}, \hat{y}, \hat{\theta})^T$.
7. Berechne die Fehlerkovarianzmatrix $\sum_{match}$.

Literatur

[1] Gutmann, J.-S.: *Robuste Navigation autonomer mobiler Systeme*. Doktorarbeit, Universität Freiburg, 2000. Kapitel 3, Seite 21-58. http://www.informatik.uni-freiburg.de/~gutmann/papers/thesis-steffen.ps.gz.

[2] Nguyen, Viet, Agostino Martinelli, Nicola Tomatis und Roland Siegwartt: *A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics*. In: *Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, IROS 2005*, Edmonton, Canada. http://asl.epfl.ch/publications/.