Generalized Iterative Closest Point

Mündliche Prüfung in der Vorlesung Autonome Roboter

bei Prof. Dr.-Ing. Michael Blaich

15.07.2024

Johannes Brandenburger, Moritz Kaltenstadler, Fabian Klimpel

Agenda

1. Einführung
2. Theorie
3. Demo: Eigene Implementierung in Python
4. Implementierung in ROS
5. Experiment
    1. Aufbau
    2. Durchführung
    3. Ergebnisse
    4. …
6. Fazit

Theorie

- Einzige wirkliche Quelle: „Generalized-ICP“ von Segal, Haehnel & Thrun (2010)
  - Ziel: Iterative-Closest-Point-Algorithmus (ICP) verbessern
  - Standard-ICP & point-to-plane in **generelles Framework** überführen
  - **Probabilistische** Betrachtung
  - Nutzung **Oberflächenstruktur** aus beiden Scans (Kovarianzmatrizen)

Theorie - Standard-ICP, point-to-plane, Generalized-ICP

- „point-to-point“ (Standard-ICP)
- „point-to-plane“
  - vergleicht Punkt mit Ebene durch Normalenvektor
- Generalized-ICP
  - quasi „plane-to-plane“
  - vergleicht die Kovarianzmatrizen der nächsten Punkte ⟶ probabilistisch
  - wenn in Ebene ⟶ Kovarianzmatrix ist „flach“

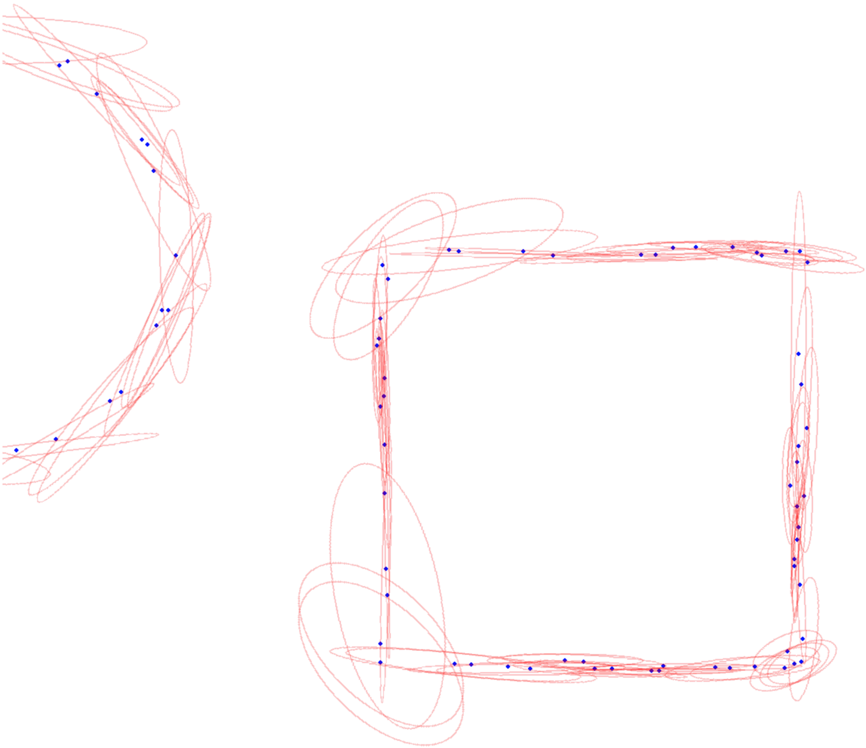

Abbildung 1: Kovarianzmatrizen (eigene Darstellung)

Theorie - Standard-ICP

- Einzige wirkliche Quelle: „Generalized-ICP“ von Segal, Haehnel & Thrun (2010)
  - Ziel: Iterative-Closest-Point-Algorithmus (ICP) verbessern
  - Standard-ICP & point-to-plane in **generelles Framework** überführen
  - **Probabilistische** Betrachtung
  - Nutzung **Oberflächenstruktur** aus beiden Scans (Kovarianzmatrizen)

Theorie - GICP Algorithmus

$T \leftarrow T_0$
while not converged **do**
    for $i \leftarrow 1$ **to** N **do**
        $m_i \leftarrow \mathrm{FindClosestPointInA}(T \cdot b_i)$
        if $\| m_i - T \cdot b_i \| \leq d_{\max}$ **then**
            $C_i^A \leftarrow \mathrm{computeCovarianceMatrix}(T \cdot b_i)$
            $C_i^B \leftarrow \mathrm{computeCovarianceMatrix}(a_i)$
        else
            $C_i^A \leftarrow 0; \quad C_i^B \leftarrow 0$
        end
        $d_i^{(T)} \leftarrow b_i - T \cdot a_i$ // Residuum / Abstand
    end
    $T \leftarrow \arg\min_T \left\{ \sum_i {d_i^{(T)}}^T \left(C_i^B + T C_i^A T^T \right)^{-1} d_i^{(T)} \right\}$
end

Theorie

- Algorithmus (ausführlicher als Orginal):

```
T = T_0
CA = computeCovarianceMatrices(A)
while has not converged do
    for i = 1 to N do
        m_i = findClosestPointInA(T * b_i)
        c_i = computeCovarianceMatrix(T * b_i)
        w_i = computeWeightMatrix(c_i, CA_i)
    end
    T = optimizeLossFunction(T, A, B, W)
end
```

Demo: Eigene Implementierung in Python

- Paper sehr mathematisch
- Algorithmus wurde nie komplett gezeigt
- zwar Implementierungen auf GitHub, aber nicht wirklich lesbar

- daher eigene Implementierung - vor allem für Verständnis
- eigene **2D-GICP-Funktion**
  - Input: Punktwolken A und B, …
  - Output: Transformationsmatrix T, …
- Version 1:
  - Visualisierung mit generierten Input-Wolken
  - iterativ durch die Steps klicken
- Version 2:
  - Simulation eines Roboters mit LiDAR-Sensor
  - Live-Berechnung der Transformation + Visualisierung

$\rightarrow$ *CODE OVERVIEW*

$\rightarrow$ *LIVE DEMO*