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) ⟶ **plane-to-plane**

Theorie - Mathematische Grundlagen

Kovarianzmatrix

- beschreibt die Streuung von Zufallsvariablen
- für Punkte in Punktwolken: Verteilung der Punkte in der Umgebung

Maximum Likelihood Estimation (MLE)

- Schätzverfahren für Parameter von Wahrscheinlichkeitsverteilungen
- der Paramter wird ausgewählt, der die beobachteten Daten am wahrscheinlichsten macht
- oft verwendet um: $\arg\max_p \ldots$ / $\arg\min_p \ldots$ zu finden

Theorie - Standard-ICP

- **Iterative Closest Point** (ICP) ist ein Algorithmus, um die Transformation zwischen zwei Punktwolken zu schätzen
- vergleicht korrespondierende Punkte in beiden Wolken
- minimiert die quadratischen Abstände korrespondierender Punkte

$T \leftarrow T_0$
while not converged **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow \texttt{FindClosestPointInA}(T \cdot b_i)$
    if $\| m_i - b_i \| \leq d_{\max}$ **then**
      $w_i \leftarrow 1$
    else
      $w_i \leftarrow 0$
    end
  end
  $T \leftarrow \arg\min_T \left\{ \sum_i w_i (\| T \cdot b_i - m_i \|)^2 \right\}$
end

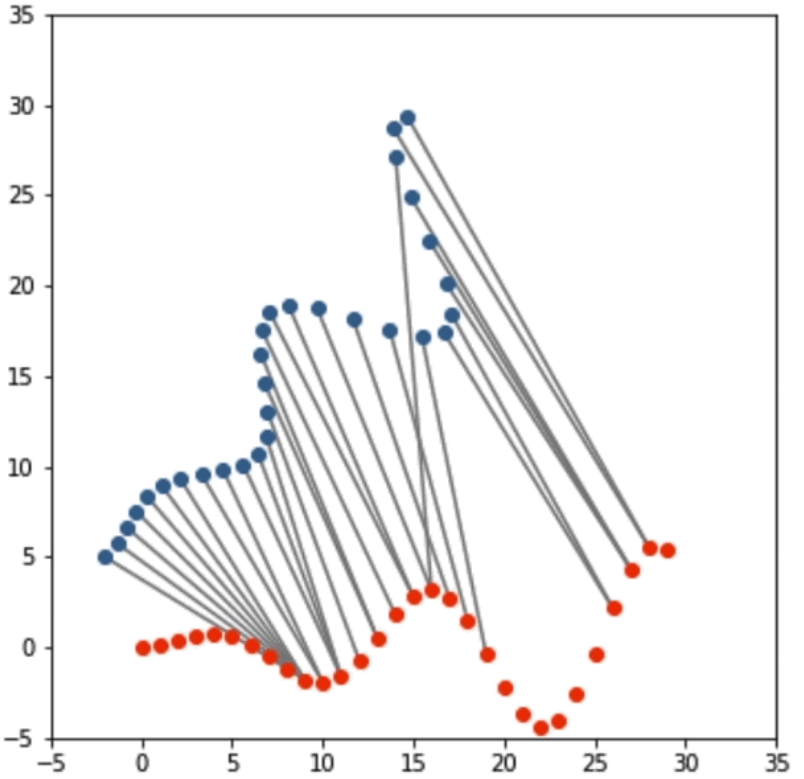

Abbildung 1: Standard-ICP (Igor Bogoslavskyi, 2021)

Theorie - Point-to-Plane-ICP

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“

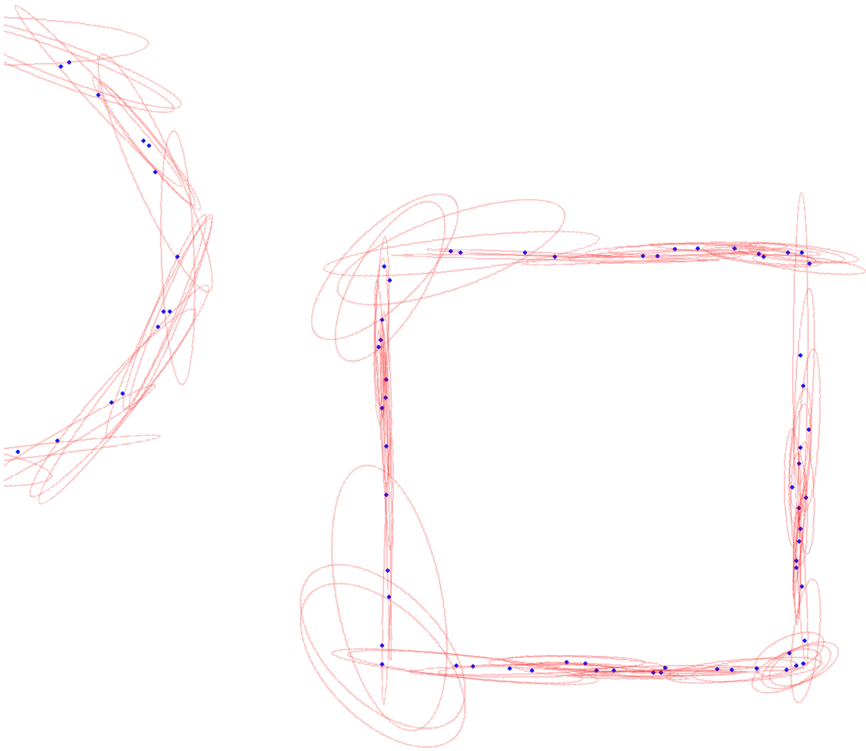

Abbildung 2: Kovarianzmatrizen (eigene Darstellung)

Theorie - GICP-Algorithmus

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

Theorie - GICP-Algorithmus

Variationen für Kovarianzmatrizen

$C_i^A \leftarrow$ `computeCovarianceMatrix`$(T \cdot b_i)$
$C_i^B \leftarrow$ `computeCovarianceMatrix`(m_i)

- für **Standard-ICP** (point-to-point):
  - $C_i^A \leftarrow 0$
  - $C_i^B \leftarrow 1$ $\longrightarrow$ keine Oberflächenstruktur berücksichtigt (einfache Gewichtung)
- für **point-to-plane**:
  - $C_i^A \leftarrow 0$
  - $C_i^B \leftarrow P_i^{-1}$ $\longrightarrow$ P_i ist die Projektionsmatrix auf die Ebene (beinhaltet Normalenvektor)
- für **plane-to-plane** (im Paper vorgeschlagene Methode):
  - `computeCovarianceMatrix` berechnet Kovarianzmatrix unter Betrachtung der nächsten 20 Punkte
    - verwendet **PCA** (Principal Component Analysis/Hauptkomponentenanalyse)

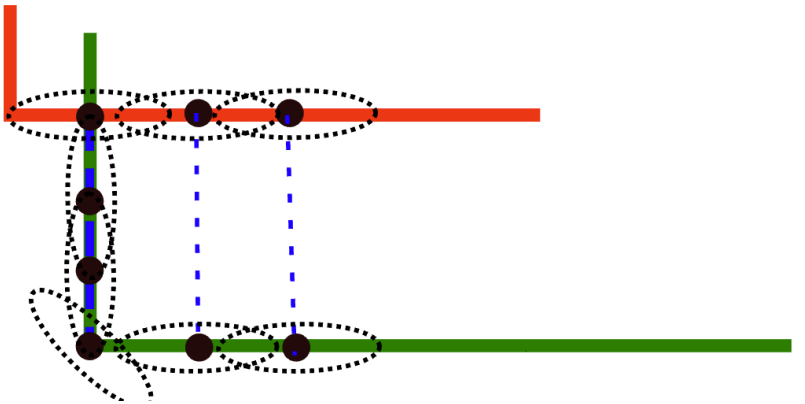

Abbildung 3: Plane-to-plane (Segal et al., 2009)

Theorie - GICP-Algorithmus

Paper Ergebnisse (Segal et al., 2009)

- GICP **genauer** bei simulierten und realen Daten
- immer noch relativ schnell und einfach
- Nutzen von Oberflächenstruktur **minimiert Einfluss von falschen Korrespondenzen**
- Parameter-Wahl für $d_{\max}$ nicht mehr so kritisch $\longrightarrow$ leichter einsetzbar in **unterschiedlichen Szenarien**

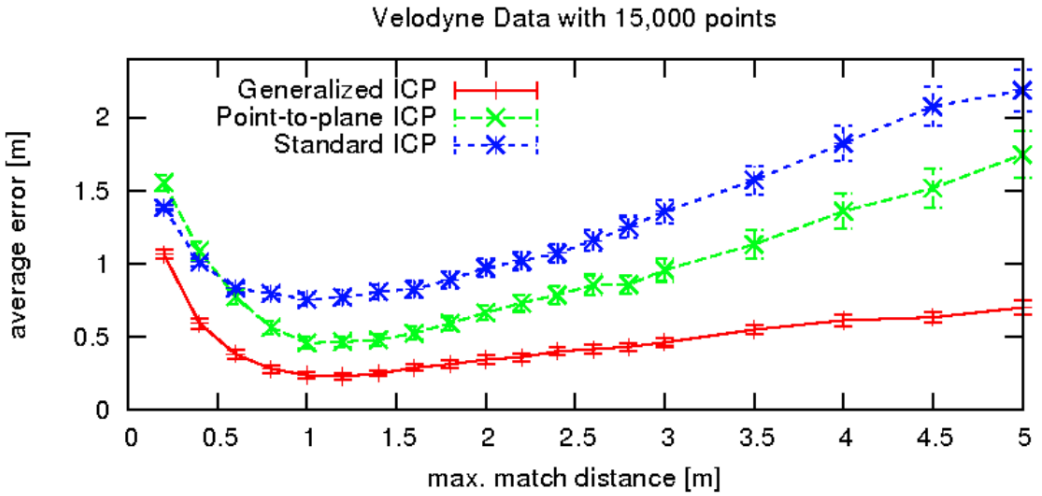

Abbildung 4: Durchschnittsfehler als Funktion von $d_{\max}$ (Segal et al., 2009)

Demo: Eigene Implementierung in Python

- Paper sehr mathematisch
- 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

→ *LIVE DEMO*

→ *CODE OVERVIEW*

Parameterisierung

```
//name des odom topics
this->declare_parameter("odom_topic", "");
//name des icp topics
this->declare_parameter("gicp_result_topic", "");
//name des zeitmessung topics
this->declare_parameter("alignment_time_topic", "");
//parameter ob gicp oder icp verwendet wird
this->declare_parameter("gicp", false);
//ob manuelle transformation gepublished wird
this->declare_parameter("publish_tf", false);

//icp parameter
this->declare_parameter("max_correspondence_distance", 0.0);
this->declare_parameter("maximum_iterations", 0);
this->declare_parameter("transformation_epsilon", 0.0);
this->declare_parameter("euclidean_fitness_epsilon", 0.0);
```

Parameterisierung über YAML file

```yaml
gicp_lio:
  ros__parameters:
    gicp: True
    publish_tf: False
    alignment_time_topic: "galignment_time"
    odom_topic: "glidar_odom"
    gicp_result_topic: "glidar_odom_eval"
    max_correspondence_distance: 0.2
    maximum_iterations: 100
    transformation_epsilon: 0.000000001
    euclidean_fitness_epsilon: 0.00001
```

```yaml
gicp_lio:
  ros__parameters:
    gicp: False
    publish_tf: False
    alignment_time_topic: "alignment_time"
    odom_topic: "lidar_odom"
    gicp_result_topic: "lidar_odom_eval"
    max_correspondence_distance: 0.2
    maximum_iterations: 100
    transformation_epsilon: 0.000000001
    euclidean_fitness_epsilon: 0.00001
```

Implementierung in Ros

- ICP:

```
pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
icp.setInputSource(src);
icp.setInputTarget(tgt);
pcl::PointCloud<pcl::PointXYZ>::Ptr output(new pcl::PointCloud<pcl::PointXYZ>);
icp.align(*output);
```

- GICP:

```
pcl::GeneralizedIterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> gicp;
gicp.setInputSource(src);
gicp.setInputTarget(tgt);
pcl::PointCloud<pcl::PointXYZ>::Ptr output(new pcl::PointCloud<pcl::PointXYZ>);
gicp.align(*output);
```

Implementierung in Ros - Problem

```
// reduce tick speed in topic_callback
tick++;
if (tick % 3 != 0) {
  return;
}
```

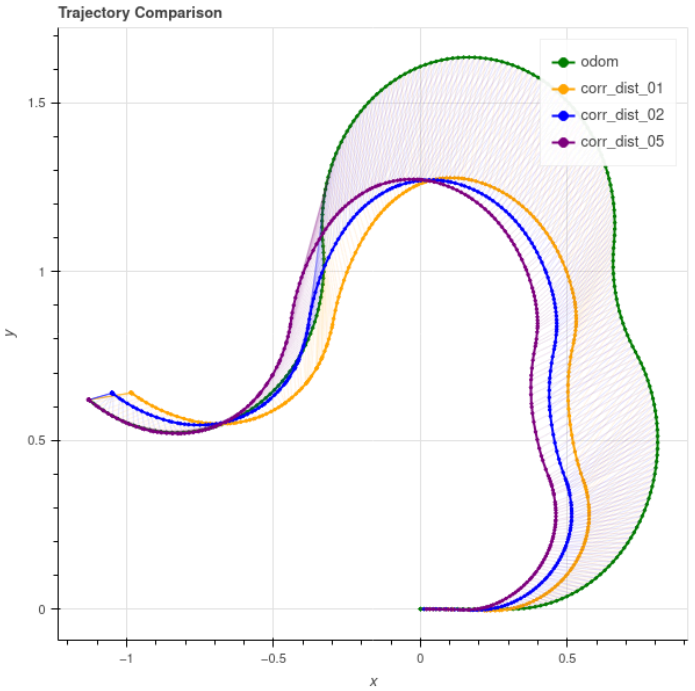

Abbildung 5: Trajectory plot with higher tick speed

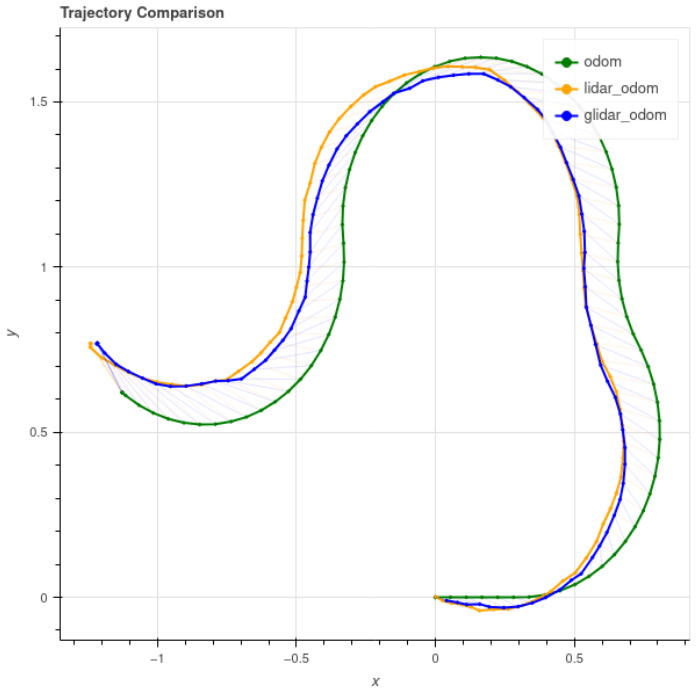

Abbildung 6: Trajectory plot with lower tick speed

Implementierung in Ros

```
auto start = std::chrono::high_resolution_clock::now();
icp.align(*output);
transformation = icp.getFinalTransformation();
auto finish = std::chrono::high_resolution_clock::now();

std::chrono::duration<double> elapsed = finish - start;
std_msgs::msg::Float64 time_msg;
time_msg.data = elapsed.count();
time_publisher_->publish(time_msg);
```

3 unterschiedliche Maps

- jeweils Bild reinmachen
- roboter war immer der gleiche

Versuchsaufbau

- fünf Durchgänge mit Standardparameterisierung (ICP & GICP)
- drei unterschiedliche Maps
- unterschiedliche Parameterisierung

(Bild-)Quellen

Igor Bogoslavskyi. (2021). https://nbviewer.org/github/niosus/notebooks/blob/master/icp.ipynb

Segal, A. V., Hähnel, D., & Thrun, S. (2009). Generalized-ICP. *Robotics: Science and Systems*. https://api.semanticscholar.org/CorpusID:231748613