

Generalized Iterative Closest Point

Mündliche Prüfung in der Vorlesung Autonome Roboter

bei Prof. Dr.-Ing. Michael Blaich



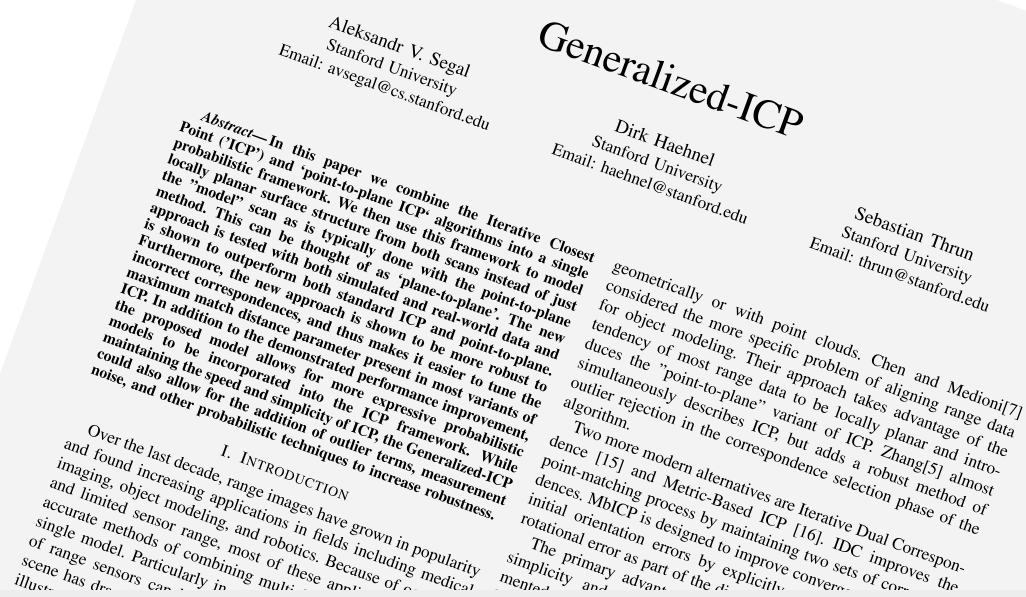
Johannes Brandenburger, Moritz Kaltenstadler, Fabian Klimpel

Agenda

1. Einführung
2. Theorie
 1. Mathematische Grundlagen
 2. Standard-ICP
 3. Point-to-Plane-ICP
 4. Generalized-ICP
 5. Ergebnisse von Segal et al.
3. Demo: Eigene Implementierung in Python
4. Implementierung in ROS - Versuch
 1. Versuchsaufbau
 2. Parameterisierung
 3. Implementierung ICP & GICP
 4. Problem
 5. Zeitmessung
 6. Maps
5. Auswertung
6. Fazit
7. (Bild-)Quellen

Einführung

- ICP: **Iterative Closest Point**
 - Scan-Matching-Algorithmus
 - Schätzung der Transformation zwischen zwei Punktwolken
 - Anwendung in der Lokalisierung mit z.B. LiDAR-Sensoren
- GICP: **Generalized-ICP**
 - veröffentlicht von Segal, Haehnel & Thrun (2009)
 - Stanford University
 - Ziel: ICP-Algorithmus verbessern und verallgemeinern
 - Standard-ICP & point-to-plane in **generelles Framework** überführen
 - **probabilistische** Betrachtung
 - Nutzung **Oberflächenstruktur** aus beiden Scans (Kovarianzmatrizen) → **plane-to-plane**



Speaker Notes

- ICP Verfahren schon in Vorlesung
 - grober Überblick
 - ICP steht für Iterative Closest Point
 - Scan-Matching-Algorithmus
 - Schätzung der Transformation zwischen zwei Punktwolken
 - Anwendung in der Lokalisierung mit z.B. LiDAR-Sensoren
 - wurde 2009 von Segal, Haehnel & Thrun verbessert
 - an Stanford University Verfahren entwickelt
 - Generalized-ICP
 - Ziel: ICP-Algorithmus verbessern und verallgemeinern
 -
- Probleme Standard-ICP:
- nicht sehr robust
 - sehr empfindlich gegenüber der Parameterwahl
 - daher schlecht in mehreren Szenarien einsetzbar
- Standard-ICP & point-to-plane in generelles Framework überführen
 - 2 große Änderungen:
 - Wahrscheinlichkeitstheorie
 - Nutzung planarer Strukturen
 - Punktwolke sind nicht random im Raum verteilt
 - haben Struktur zb von Wänden
 - GICP nutzt diese Struktur in Form von Kovarianzmatrizen
 - Überleitung Kovarianzmatrizen -> Mathematische Grundlagen

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 Parameter wird ausgewählt, der die beobachteten Daten am wahrscheinlichsten macht
- oft verwendet um: $\arg \max_p \dots$ / $\arg \min_p \dots$ zu finden

Speaker Notes

- Mathematische Grundlagen sind lediglich für ein gemeinsames Verständnis
- sodass auch Leute, die nicht die Vorlesung besucht haben, den Vortrag verstehen könnten
- Kovarianzmatrix beschreibt die Streuung von Zufallsvariablen
- in unserem Kontext: Verteilung von Punkte in der Umgebung
- wo unsere Punkte mit welcher Wahrscheinlichkeit liegen
- Maximum Likelihood Estimation brauchen wir für die Schätzung der Transformation bei ICP und GICP
- Schätzverfahren für Parameter von Wahrscheinlichkeitsverteilungen
- versucht quasi die Parameter zu finden, die eine gegebene Wahrscheinlichkeitsverteilung am besten beschreiben

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

```

1  $T \leftarrow T_0$ 
2 while not converged do
3   for  $i \leftarrow 1$  to  $N$  do
4      $m_i \leftarrow \text{FindClosestPointInA}(T \cdot b_i)$ 
5     if  $\| m_i - b_i \| \leq d_{\max}$  then
6        $w_i \leftarrow 1$ 
7     else
8        $w_i \leftarrow 0$ 
9     end
10  end
11   $T \leftarrow \arg \min_T \left\{ \sum_i w_i (\| T \cdot b_i - m_i \|^2) \right\}$ 
12 end

```

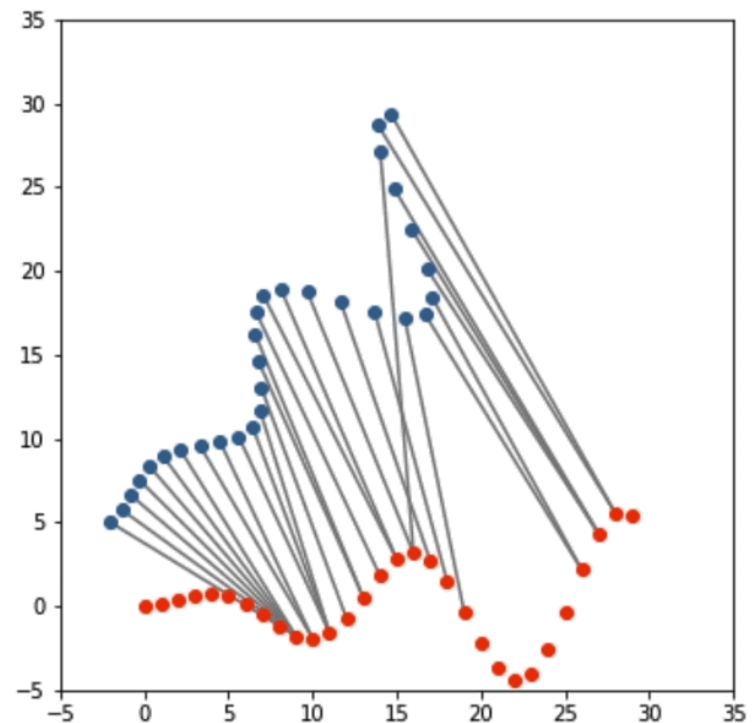


Abbildung 1: Standard-ICP (Igor Bogoslavskyi, 2021)

Speaker Notes

- wie schon gesagt: Standard ICP lässt uns Transformation zwischen zwei Punktwolken schätzen
- dabei ist er sehr einfach und schnell
- vergleicht korrespondierende Punkte in beiden Wolken
- minimiert die quadratischen Abstände korrespondierender Punkte
- schätzt so die Transformation
- in Pseudocode dargestellt
- Startwert für Transformation T_0
 - kann bereits eine sinnvolle Schätzung sein (z.b. Odometrie)
- Loop bis der Algorithmus konvergiert (daher auch „Iterative“)
- für jeden Punkt in der Sourcewolke b_i wird der nächste Punkt in der Targetwolke A gesucht
- dann wird geschaut ob der Abstand kleiner als das Threshold d_{\max} ist
 - Parameter steuert also, welche Punkte berücksichtigt werden und welche nicht
 - ist je nach Anwendungsszenario unterschiedlich
 - wenn Roboter schneller fährt, dann muss d_{\max} größer sein
 - schwierig einzustellen
- Punkt wird gewichtet, oder nicht
- am Ende jedes Durchlaufs wird die Transformation berechnet
 - durch Veränderung der Transformationsparameter
 - so dass die quadratischen Abstände minimiert werden

Theorie - Point-to-Plane-ICP

- **Point to Plane ICP** ist eine Erweiterung des ICP Algorithmus
- vergleicht korrespondierende Punkte in einer Wolke zu Ebenen in der anderen
- Ebenen wird durch Punkt und Normalenvektor definiert

$$T \leftarrow \arg \min_T \left\{ \sum_i ((T \cdot b_i - m_i) \cdot \mathbf{n}_i)^2 \right\}$$

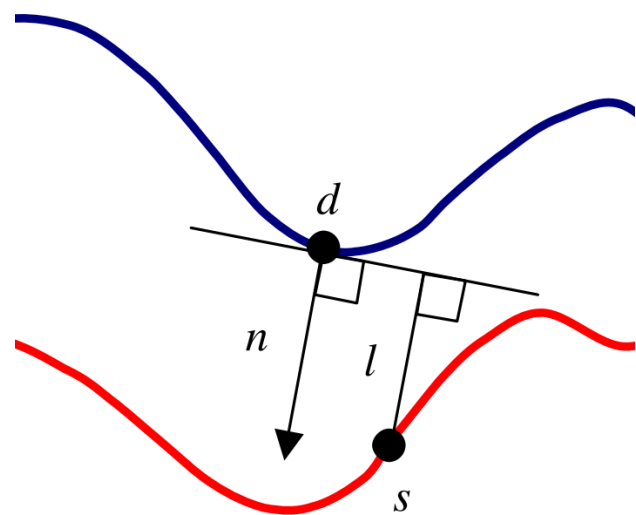


Abbildung 2: Point-to-Plane-ICP (Kok-Lim Low, 2004)

Speaker Notes

- Point-to-Plane-ICP ist eine Erweiterung des Standard-ICP
- Standard-ICP betrachtet keine Oberflächenstruktur
 - bei Laserscan zum Beispiel:
 - wenn 2 mal eine Wand gescant
 - Punkte zwar durch die Abtaste unterschiedlichen Stellen im Raum
 - kann sein, dass die Bewegung zur Wand nicht wirklich groß war
- um dem entgegenzuwirken, vergleicht Point-to-Plane-ICP Punkte in einer Wolke zu Ebenen in der anderen
- Ebenen wird durch Punkt und Normalenvektor definiert
- Optimierungsfunktion in der letzten Zeile des Algorithmus:
 - minimiert quadratische Abstände zwischen Punkt und Ebene

Theorie - Standard-ICP, Point-to-Plane, Generalized-ICP

- **Point-to-Point**
 - Standard-ICP
 - vergleicht Punkt mit Punkt
- **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“

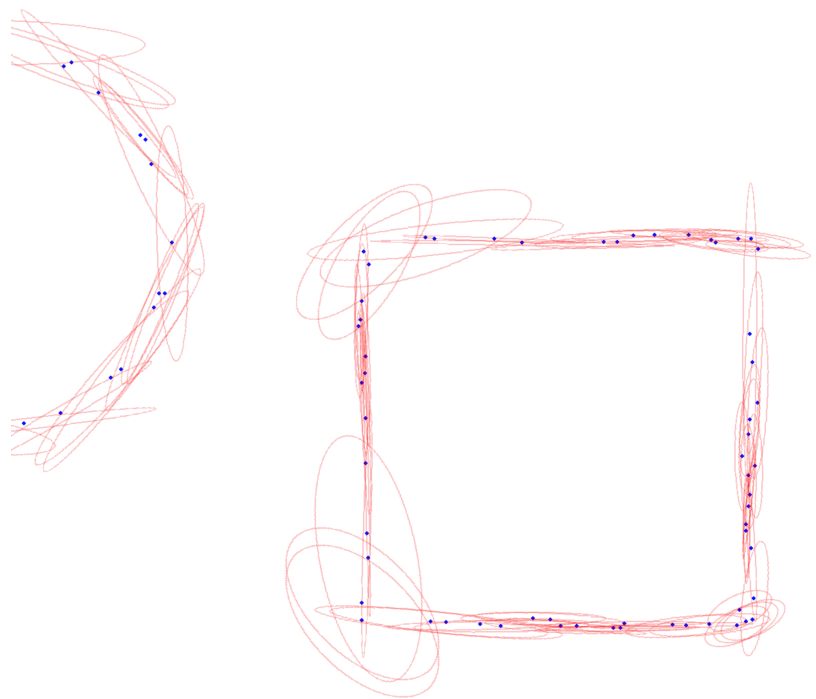


Abbildung 3: Kovarianzmatrizen (eigene Darstellung)

Speaker Notes

- hier nochmal ein Überblick über die verschiedenen ICP-Verfahren
- Point-to-Point: Standard-ICP
 - vergleicht Punkt mit Punkt
 - einfach und schnell
 - aber nicht robust sehr empfindlich gegenüber Parameterwahl
- Point-to-Plane:
 - vergleicht Punkt mit Ebene durch Normalenvektor
 - besser als Standard-ICP
 - nutzt Oberflächenstruktur einer Punktwolke
 - aber eben nur von einer
- Generalized-ICP:
 - quasi „Plane-to-Plane“
 - vergleicht die Kovarianzmatrizen der nächsten Punkte → probabilistisch
 - wenn in Ebene → Kovarianzmatrix ist „flach“
 - sieht man in Bild rechts
 - nutzt Oberflächenstruktur beider Punktwolken
 - welche Ergebnisse dies hat, dazu später mehr

Theorie - GICP-Algorithmus

```
1  $T \leftarrow T_0$ 
2 while not converged do
3   for  $i \leftarrow 1$  to  $N$  do
4      $m_i \leftarrow \text{FindClosestPointInA}(T \cdot b_i)$ 
5      $d_i^{(T)} \leftarrow b_i - T \cdot m_i$  // Residuum / Abstand
6     if  $\|d_i^{(T)}\| \leq d_{\max}$  then
7        $C_i^A \leftarrow \text{computeCovarianceMatrix}(T \cdot b_i)$ 
8        $C_i^B \leftarrow \text{computeCovarianceMatrix}(m_i)$ 
9     else
10       $C_i^A \leftarrow 0$ ;  $C_i^B \leftarrow 0$ 
11    end
12  end
13   $T \leftarrow \arg \min_T \left\{ \sum_i d_i^{(T)^T} (C_i^B + T C_i^A T^T)^{-1} d_i^{(T)} \right\}$  // Maximum Likelihood
    Estimation
14 end
```

Speaker Notes

- im Paper wurde Algorithmus nie zusammenhängend dargestellt
 - deshalb hier nochmals selber zusammengebaut
- Anfang des Algorithmus gleich wie bei Standard-ICP
- statt Gewichtung mit 0 oder 1 werden Kovarianzmatrizen verwendet
 - wie genau diese berechnet bzw gewählt werden, dazu mehr auf der nächsten Folie
- Minimierungsfunktion am Ende des Schleifendurchlaufs anders
 - nutzt Gewichtungsmatrix, die aus den Kovarianzmatrizen berechnet wird
 - mittlere Teil der Minimierungsfunktion
- im Paper wird für die Optimierung der Transformation also für $\arg \min$ Maximum Likelihood Estimation verwendet
 - wählt Transformationsmatrix T so dass die Verteilung am wahrscheinlichsten ist

Theorie - GICP-Algorithmus

Variationen für Kovarianzmatrizen

$C_i^A \leftarrow \text{computeCovarianceMatrix}(T \cdot b_i)$
 $C_i^B \leftarrow \text{computeCovarianceMatrix}(m_i)$

- für **Standard-ICP** (Point-to-Point):
 - $C_i^A \leftarrow 0$
 - $C_i^B \leftarrow 1$ → keine Oberflächenstruktur berücksichtigt (einfache Gewichtung)
- für **Point-to-Plane**:
 - $C_i^A \leftarrow 0$
 - $C_i^B \leftarrow P_i^{-1}$ → 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)

```

1   $T \leftarrow T_0$ 
2  while not converged do
3    for  $i \leftarrow 1$  to  $N$  do
4       $m_i \leftarrow \text{FindClosestPointInA}(T \cdot b_i)$ 
5       $d_i^{(T)} \leftarrow b_i - T \cdot m_i$  // Residuum / Abstand
6      if  $\|d_i^{(T)}\| \leq d_{\max}$  then
7         $C_i^A \leftarrow \text{computeCovarianceMatrix}(T \cdot b_i)$ 
8         $C_i^B \leftarrow \text{computeCovarianceMatrix}(m_i)$ 
9      else
10        $C_i^A \leftarrow 0$ ;  $C_i^B \leftarrow 0$ 
11    end
12  end
13   $T \leftarrow \arg \min_T \left\{ \sum_i d_i^{(T)^T} (C_i^B + T C_i^A T^T)^{-1} d_i^{(T)} \right\}$  // Maximum Likelihood Estimation
14 end

```

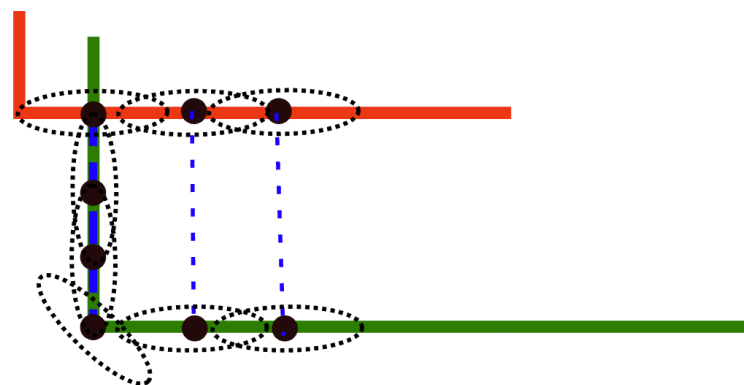


Abbildung 4: Plane-to-Plane (Segal et al., 2000)

Speaker Notes

- Wahl der Kovarianzmatrizen sind also entscheidend für den GICP-Algorithmus
- auch der Grund, warum „Generalized“ ICP
 - denn es lässt sich auch Standard-ICP und Point-to-Plane-ICP dadurch abdecken
- für Standard-ICP werden Kovarianzmatrizen einfach auf 0 bzw 1 gesetzt
 - dadurch werden die Punkte einfach gewichtet und es wird keine Oberflächenstruktur berücksichtigt
- für Point-to-Plane-ICP werden die Source-Kovarianzmatrizen auf Projektionsmatrizen gesetzt
 - diese beinhalten den Normalenvektor der Ebene
 - Oberflächenstruktur der einen Wolke berücksichtigt
- aber richtig gut erst bei dem im Paper vorgeschlagenen Verfahren
 - quasi „Plane to Plane“
- hier werden wirklich Kovarianzmatrizen ausgerechnet
 - 20 umliegende Punkte werden betrachtet
 - Verteilung mit Hauptkomponentenanalyse bestimmt
 - wenn hier eine genauere, mathematische Erklärung gewünscht, später darauf eingehen
- allerdings auch etwas mehr Rechenaufwand bei jeder Iteration
- Berechnen Kovarianzmatrizen geschieht bei beiden Wolken -> Berücksichtigung beider Oberflächenstrukturen
- Bild rechts zeigt Kovarianzmatrizen für paar Punkte
 - man sieht:
 - Ausrichtung/Wölbung stimmt mit Ebene überein
 - Kovarianzmatrix ist „flach“

Theorie - GICP-Algorithmus

Ergebnisse von Segal et al.

- 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 → leichter einsetzbar in **unterschiedlichen Szenarien**

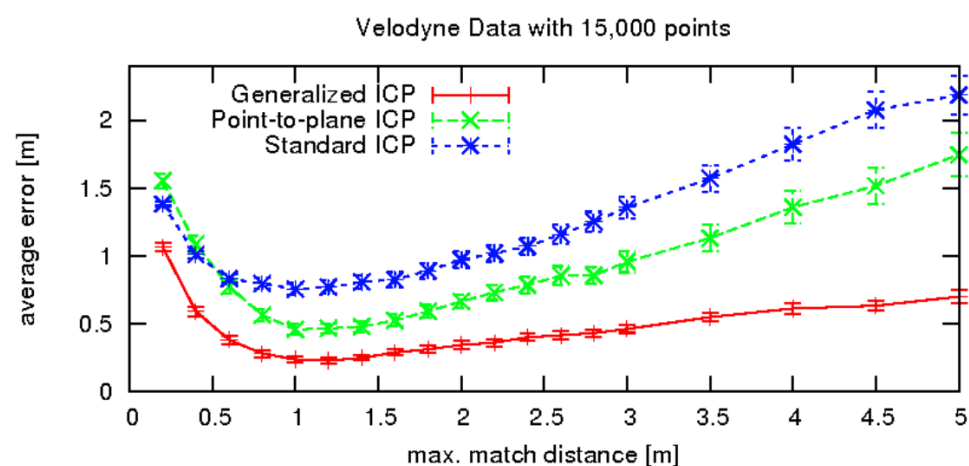


Abbildung 5: Durchschnittsfehler als Funktion von d_{\max} (Segal et al., 2009)

Speaker Notes

- insgesamt GICP robuster und genauer bei simulierten und realen Daten
- dabei immer noch relativ schnell und einfach
 - auch wenn mehr Rechenaufwand durch Kovarianzmatrizen
- Nutzen von Oberflächenstruktur minimiert Einfluss von falschen Korrespondenzen, was bei Standard-ICP ein Problem war
- dadurch auch Parameter-Wahl für d_{\max} nicht mehr so kritisch
 - leichter einsetzbar in unterschiedlichen Szenarien
- in Bild unten
 - Durchschnittsfehler als Funktion von d_{\max}
 - Generalized-ICP ganz unten, hat am die niedrigsten Fehler

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*

Speaker Notes

Version 1

- Visualisierung mit generierten Input-Wolken
- Source und Target Punktwolke
 - Quadrat und Kreis als Punktwolken
 - Beide Punktwolken haben unterschiedliches Rauschen
 - Unterschiedlich viele Punkte
 - Keine Sortierung
- Kovarianzmatrizen werden dargestellt
- Bissle durchsteppen

Version 2

- Bissle mitm Roboter rumfahren
 - Einmal um Kreis
- => schlechte Rotation estimation
- Einmal an gerader Wand entlang
- => schlechte Translation estimation

Demo: Eigene Implementierung in Python

Code Overview - GICP

GICP-Funktion:

```
def gicp(source_points, target_points, max_iterations=100, tolerance=1e-6,
        max_distance_correspondence=150, max_distance_nearest_neighbors=50):
    for iteration in range(max_iterations):
        # Calculate covariance matrices for weight matrices
        # Will be used in loss function
        ...

        # Minimize the loss function
        transformation = scipy.optimize.fmin_cg(
            f=loss,
            x0=transformation,
            fprime=grad_loss)

        # Check for convergence
        if delta_loss < tolerance:
            break

    return transformation
```

Demo: Eigene Implementierung in Python

Code Overview - Roboter

GICP-Aufruf:

```
transformation_matrix = gicp(  
    _source_points,  
    _target_points,  
    max_distance_nearest_neighbors=200,  
    tolerance=1,  
)
```

Demo: Eigene Implementierung in Python

Code Overview - Roboter

Berechnung der neuen Schätzung:

```
# Get delta x, y and yaw from transformation matrix
delta_x = -transformation_matrix[0, 2]
delta_y = -transformation_matrix[1, 2]
delta_yaw = -np.arctan2(transformation_matrix[1, 0], transformation_matrix[0,
0])

# Calculate estimations for new position and orientation
new_estimated_x = last_estimated_x
    + delta_x * math.cos(last_estimated_yaw)
    - delta_y * math.sin(last_estimated_yaw)

new_estimated_y = last_estimated_y
    + delta_x * math.sin(last_estimated_yaw)
    + delta_y * math.cos(last_estimated_yaw)

new_estimated_yaw = last_estimated_yaw + delta_yaw
```

Implementierung in ROS - Versuch

Leitfrage

Ist Generalized-ICP besser als Standard-ICP und wie verhält sich der Algorithmus in unterschiedlichen Szenarien?

Speaker Notes

I

V

-

-

-

OS - Versuch

rung

parameterisierung (ICP & GICP)

- gewährleistet eine umfassende und nachvollziehbare Bewertung der Performance konfigurieren. OS - Ver von GICP und Yaml-Dateien fünf als Konfiguration: Testkonfigurationen: Konfigurationsdateien Deserialisierung im Yaml-Format, die die Konfigurationen der Experimente enthält. Parameter und die Konfigurationen der Experimente Szenarien: Parameter den Parameter

Implementierung in ROS - Versuch

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);
```

Speaker Notes

-max correspondence distance: maximale distanz mit der ein punkt aus der source wolke mit einem punkt aus der target wolke korrespondieren kann -
maximum iterations: maximale anzahl an iterationen die der algorithmus -transformation epsilon: die maximal zulässige quadratische Differenz zwischen zwei aufeinanderfolgenden Transformationen -
euclidean fitness epsilon: Der maximal zulässige euklidische Fehler zwischen zwei aufeinanderfolgenden Schritten in der ICP-Schleife
fehler = durschnitt aller Unterschiede zwischen den korrespondierenden Punkten

Implementierung in ROS - Versuch

Parameterisierung über YAML file

```
gicp_lio:
  ros__parameters:
    gicp: True
    publish_tf: False
    alignment_time_topic:
"alignment_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
```

```
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 - Versuch

Implementierung ICP & GICP

- 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);
```

Speaker Notes

- ICP und GICP sind beide in der PCL-Bibliothek implementiert
- beide benötigen als Input die Source- und Target-Punktwolke
- align-Funktion berechnet die Transformation
- icp.align(output): Führt den ICP-Algorithmus aus, um die beste Übereinstimmung zwischen der Quell- und Zielpunktwolke zu finden und speichert das Ergebnis in der Ausgabepunktwolke

Implementierung in ROS - Versuch

Zeitmessung

```
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);
```

Speaker Notes

- Zeitmessung wird mit der C++-Chrono-Bibliothek durchgeführt
- start und finish markieren den Anfang und das Ende der Zeitmessung
- elapsed berechnet die Dauer der Zeitmessung
- time_msg wird erstellt und die Zeit wird in die Nachricht geschrieben
- time_publisher veröffentlicht die Nachricht auf dem entsprechenden Topic
- Überprüfung ob GICP auch wirklich aufwendiger in der Berechnung
- wie ändern Parameter die Zeit

Implementierung in ROS - Versuch

Problem: Aufsummierende Fehler

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

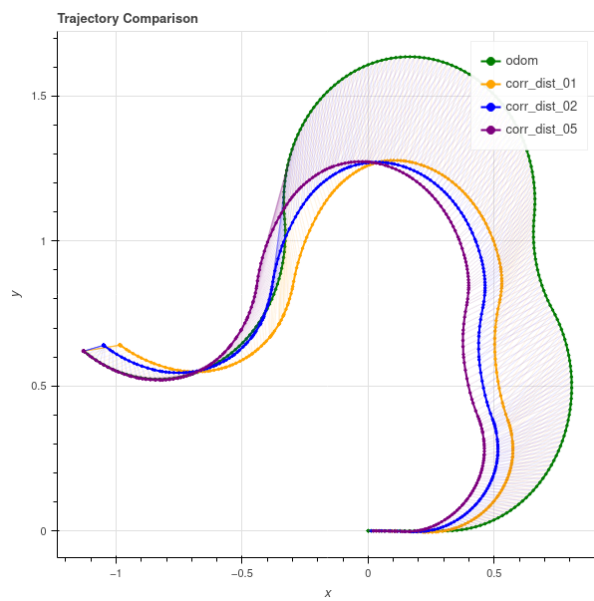


Abbildung 6: Trajectory plot with higher tick speed

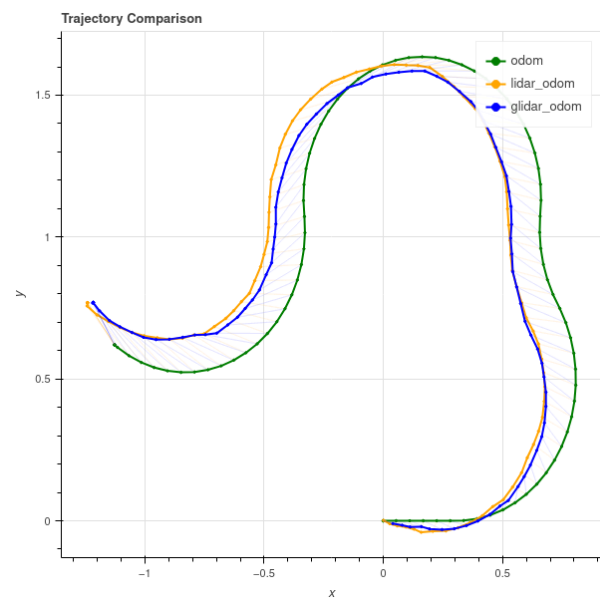


Abbildung 7: Trajectory plot with lower tick speed

Speaker Notes

- Problem: Aufsummierende Fehler
- schlechte Ergebnisse wenn jeder Tick berücksichtigt wird
- weniger Aufrufe der (G)ICP Algorithmen
- weniger Datenpunkte
- deutlich bessere Ergebnisse wenn nur jeder dritte Tick berücksichtigt wird
- rechenleistung wird gespart

Implementierung in ROS - Versuch

Turtlebot3 World

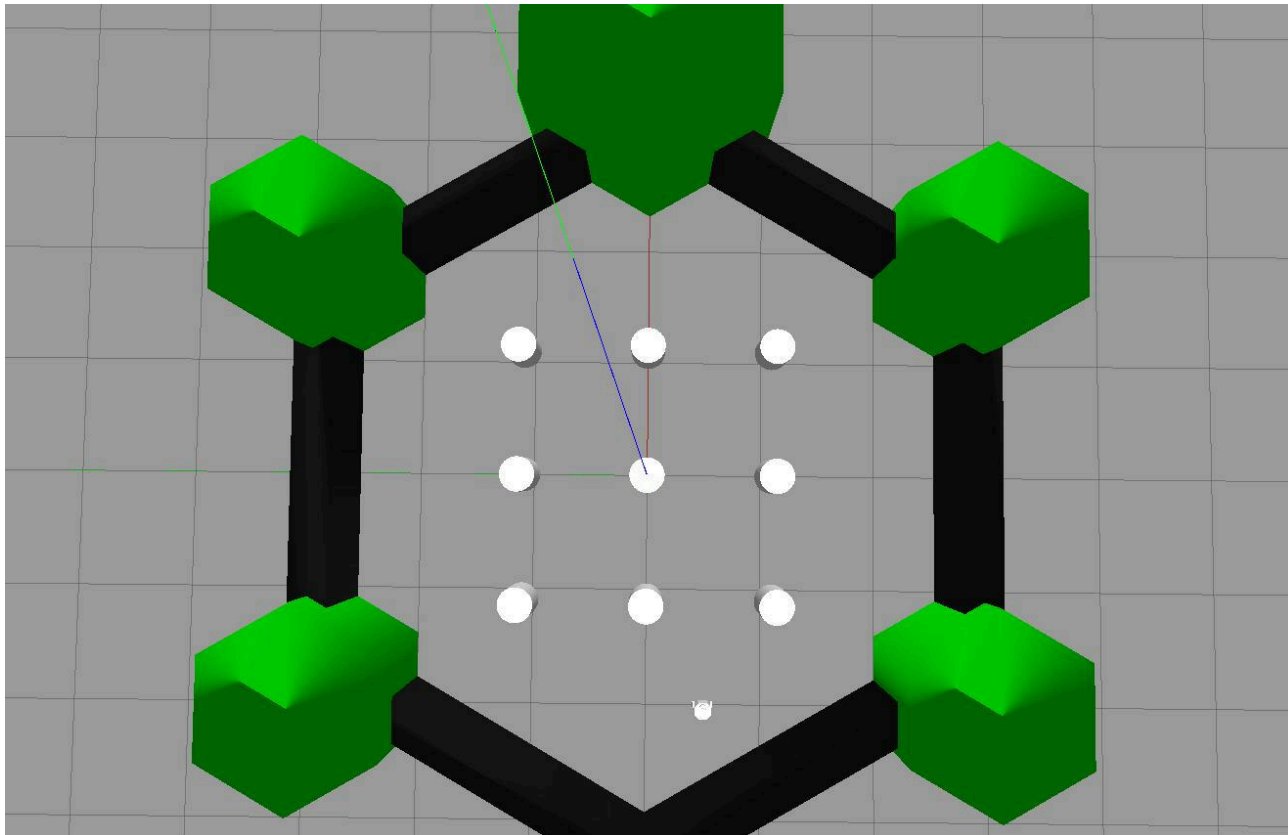


Abbildung 8: Screenshot Gazebo

Speaker Notes

- Turtlebot3 World: Standard Gazebo-Welt für Turtlebot3
- sechseck und 9 zylinder
- mittel komplexe Umgebung
- viele Datenpunkte

Implementierung in ROS - Versuch

Turtlebot3 World

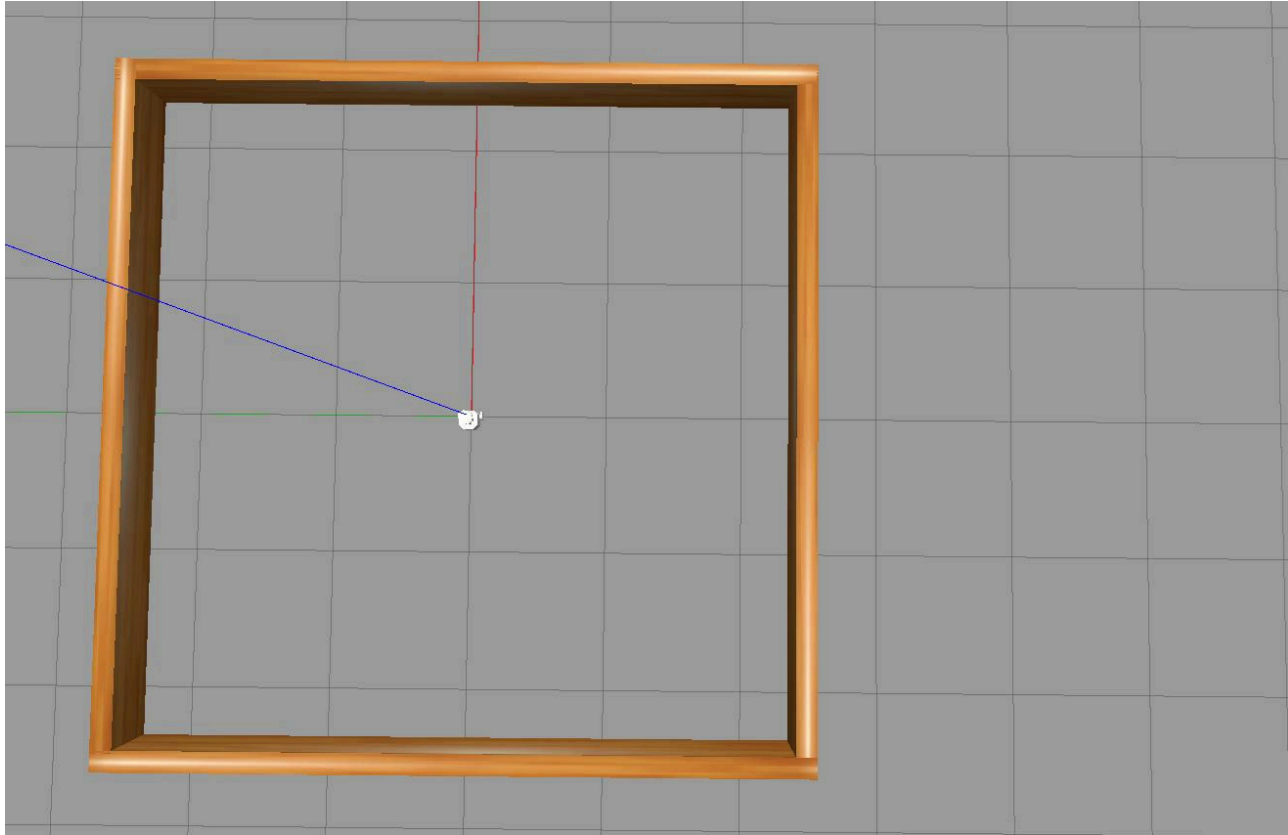


Abbildung 9: Screenshot Gazebo

Speaker Notes

- rechteck
- keine Hindernisse
- einfachste Umgebung
- wenig Datenpunkte

Implementierung in ROS - Versuch

Turtlebot3 ICP World

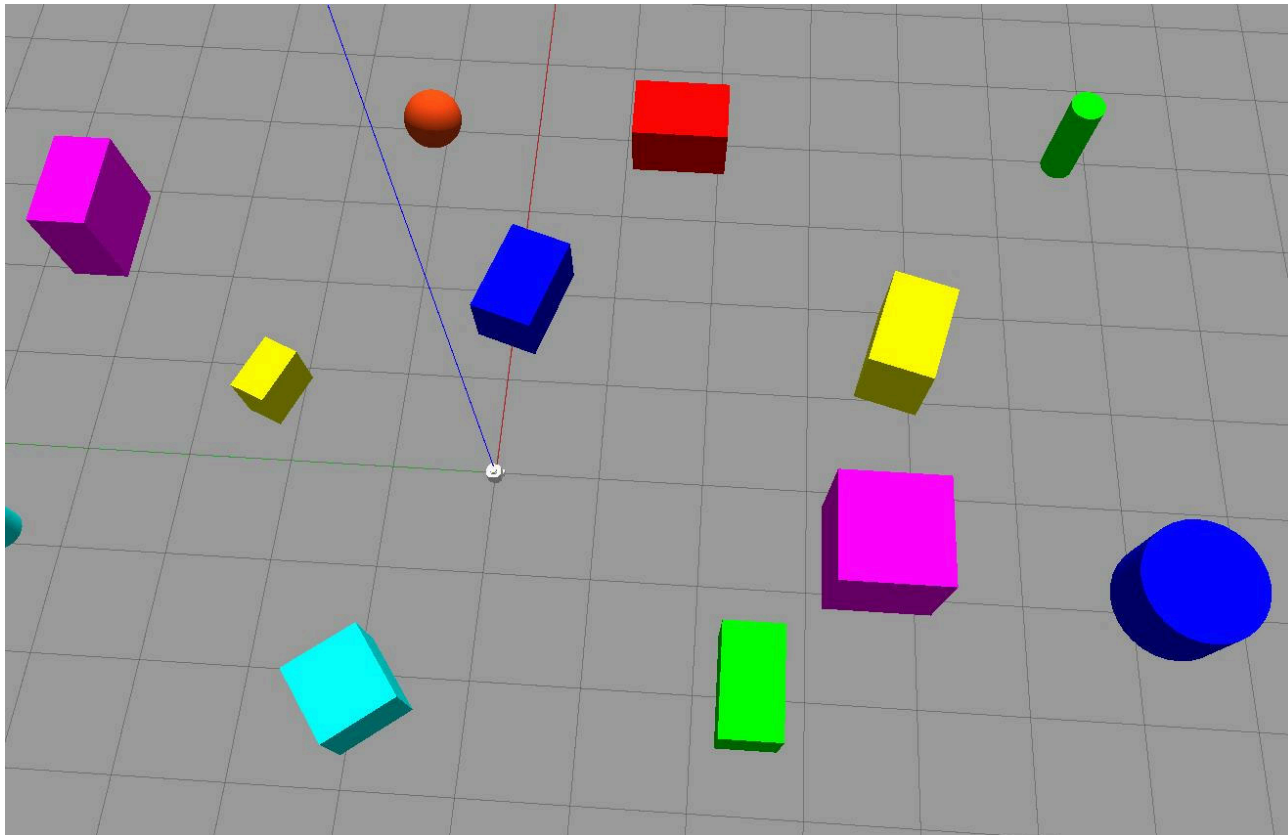


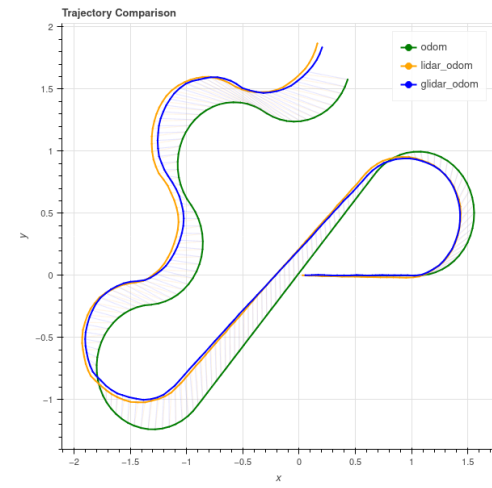
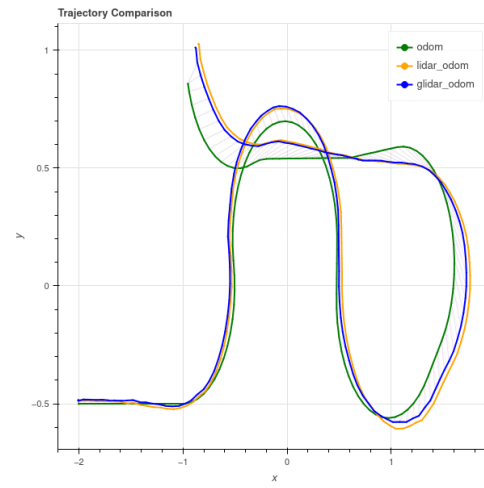
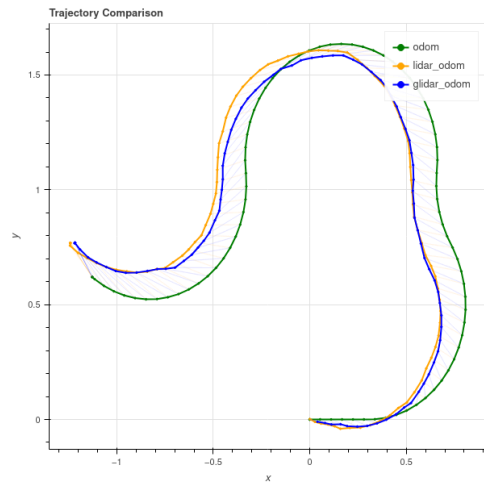
Abbildung 10: Screenshot Gazebo

Speaker Notes

- icp welt von aufgabe 2
- unterschiedlichste objekte
- keine begrenzung der umgebung
- komplexeste Umgebung
- mittel viele Datenpunkte

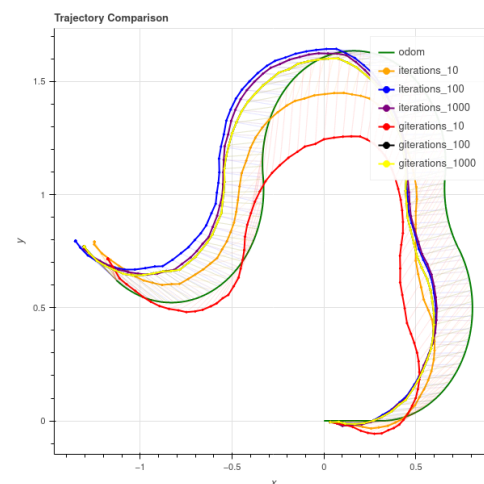
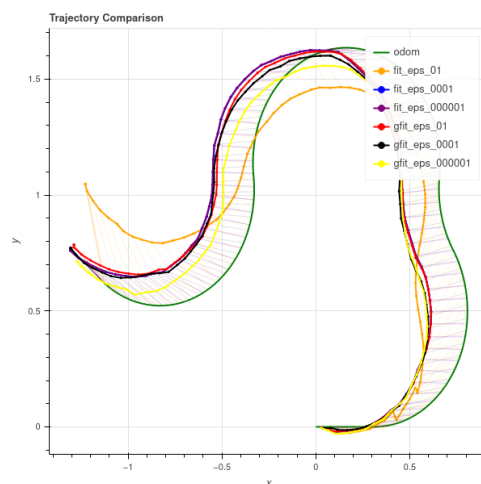
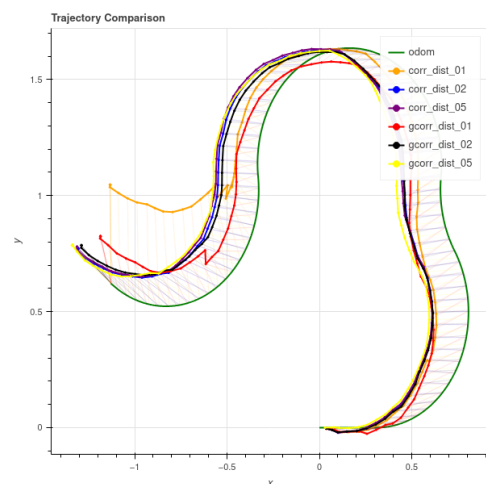
Auswertung

Drei unterschiedliche Maps



Auswertung

Unterschiedliche Parameterisierung



Speaker Notes

corr_dist: max correspondence distance

- icp mit 0,1 deutlich schlechter in position und orientation
- gicp mit 0,1 sogar am besten
- keine großen Unterschiede von 0,2 und 0,5

=> gicp robuster gegenüber Parameterwahl

- alignment time bei gicp deutlich höher
- alignment time bei gicp und icp parameterunabhängig

fit_eps: euclidean fitness epsilon

- icp mit 0,1 deutlich schlechter in position und orientation
- gicp mit 0,00001 am besten
- rest relativ ähnlich
- alignment gleich wie bei corr_dist

=> gicp robuster gegenüber Parameterwahl

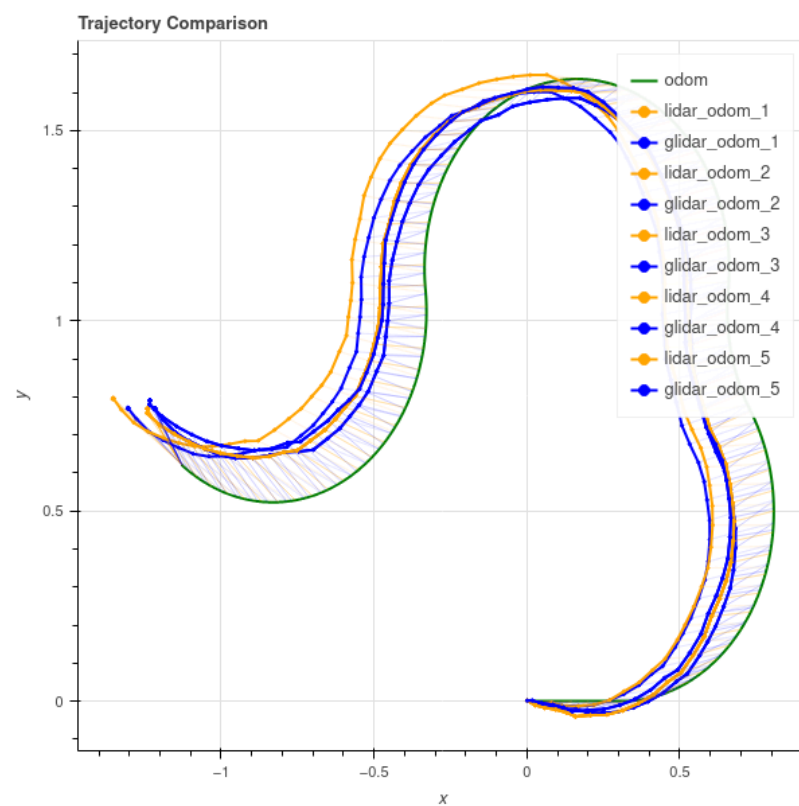
iterations: maximum iterations

- gicp mit 10 deutlich schlechter in position und orientation
- icp mit 10 deutlich schlechter in orientation
- rest relativ ähnlich

- alignment time bei gicp 10 iterations niedriger => algorithmus terminiert
- restliche Zeit bei gicp und icp kaum unterschiede

Auswertung

Fünf Durchgänge mit Standardparameterisierung (ICP & GICP)



Speaker Notes

- durchläufe relativ konstant
- gibt abweichungen in position und orientation
- gibt fast perfekt gleiche trajectories

=> Ist Simulation

Fazit

- Leitfrage: Ist Generalized-ICP besser als Standard-ICP und wie verhält sich der Algorithmus in unterschiedlichen Szenarien?
- keine deutlich besseren Ergebnisse bei GICP
- Parameter-Wahl nicht so kritisch wie bei ICP kann bestätigt werden
- keine marginalen Unterschiede zwischen ICP & GICP in den unterschiedlichen Maps
- GICP ist aufwendiger in der Berechnung
- Abhängigkeit von der Simulationsumgebung
- Unterschiede zu Paper, da 3D-Scan und deutlich mehr Punkte

(Bild-)Quellen

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

Kok-Lim Low. (2004). *Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration*. https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf

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

Anhang - Bestimmung der Kovarianzmatrizen bei Plane-to-Plane GICP

```
def compute_covariance_matrix_single_point(point, neighbors):
    epsilon = 1e-6
    covariance_ground = np.array([[epsilon, 0],
                                   [0, 1]])

    covariance = np.cov(neighbors, rowvar=False)
    U, D, V = np.linalg.svd(covariance)  # Singular Value
    Decomposition
    covariance_aligned = U @ covariance_ground @ U.T  # Align covariance matrix
    return covariance_aligned
```

Single Value Decomposition

- Zerlegung einer Matrix in drei Matrizen:
 - Rotation → Skalierung → Rotation
- wird hier verwendet, um die Rotation der Kovarianzmatrix zu bestimmen
 - Rotation sollte so sein, wie die Nachbarn liegen (Ebene)
- die Rotation wird dann auf die „Standard-Kovarianzmatrix“ angewendet
 - bildet so die Ebene ab

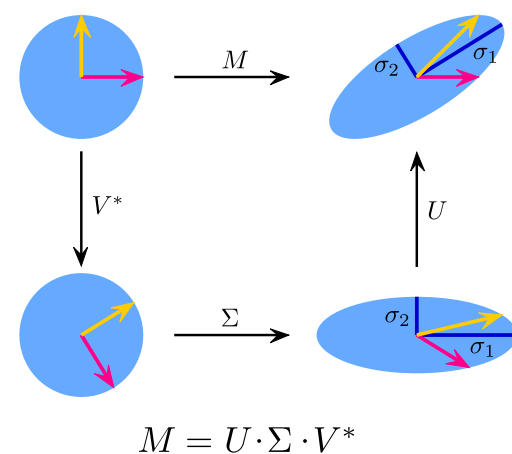


Abbildung 11: Singular Value Decomposition (Wikipedia)

Speaker Notes

- es wird Single Value Decomposition verwendet
- Zerlegung einer Matrix in drei Matrizen:
 - Rotation → Skalierung → Rotation
- wird hier verwendet, um die Rotation der Kovarianzmatrix zu bestimmen
 - Rotation sollte so sein, wie die Nachbarn liegen
- die Rotation wird dann auf die „Standard-Kovarianzmatrix“ angewendet
 - bildet so die Ebene ab
- in Paper wird in Fußnote erwähnt, dass sie bei ihrer Implementierung PCA (Hauptkomponentenanalyse) verwendet haben
- bei unserem eigene 2D-Skript hat das allerdings nicht so gut funktioniert wie die Alternative mit SVD