Generalized Iterative Closest Point

Mündliche Prüfung in der Vorlesung Autonome Roboter

bei Prof. Dr.-Ing. Michael Blaich

18.10.2024

Johannes Brandenburger, Moritz Kaltenstadler, Fabian Klimpel

Hochschule Konstanz
Fakultät Informatik

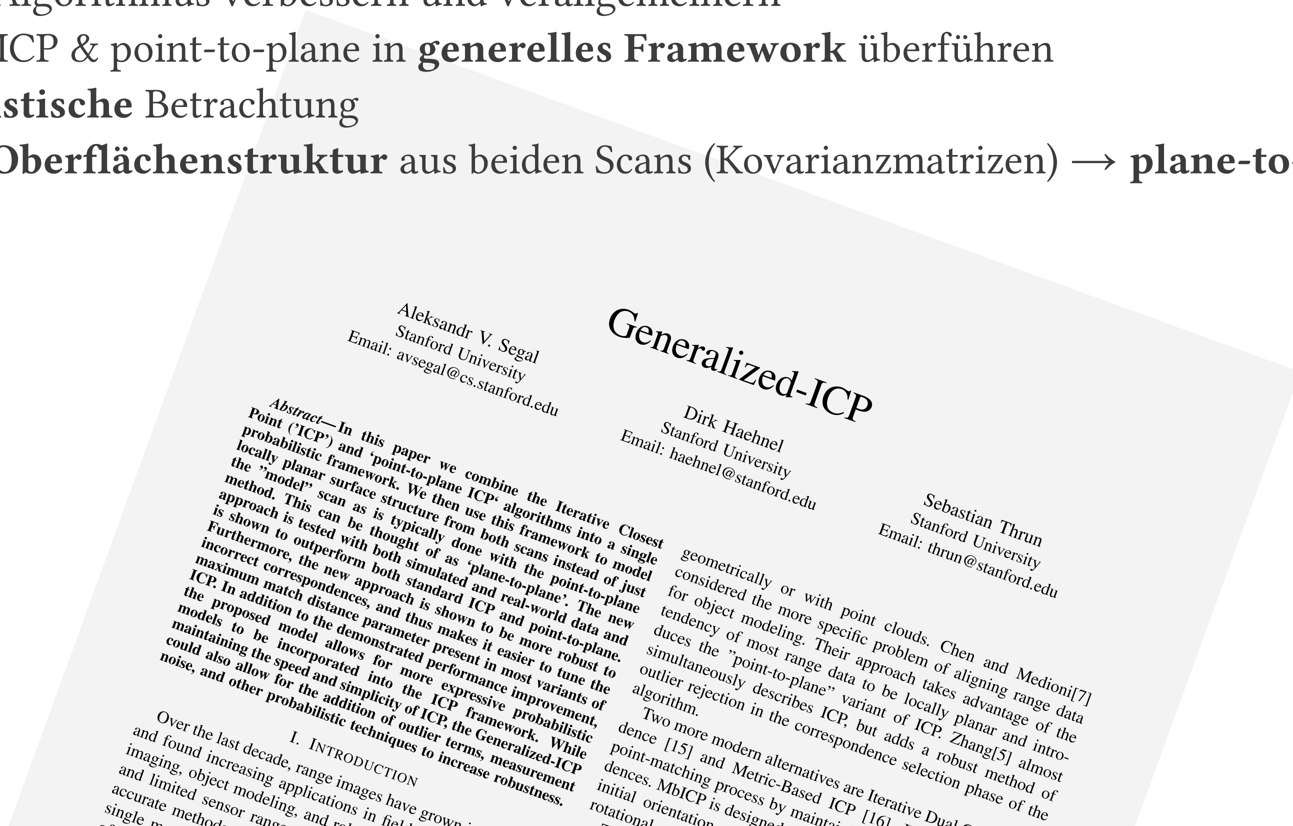
Agenda

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**



Generalized-ICP

Aleksandr V. Segal
Stanford University
Email: avsegal@cs.stanford.edu

Dirk Haehnel
Stanford University
Email: haehnel@stanford.edu

Sebastian Thrun
Stanford University
Email: thrun@stanford.edu

***Abstract*—In this paper we combine the Iterative Closest Point ('ICP') and 'point-to-plane ICP' algorithms into a single probabilistic framework. We then use this framework to model locally planar surface structure from both scans instead of just the "model" scan as is typically done with the point-to-plane method. This can be thought of as 'plane-to-plane'. The new approach is tested with both simulated and real-world data and is shown to outperform both standard ICP and point-to-plane. Furthermore, the new approach is shown to be more robust to incorrect correspondences, and thus makes it easier to tune the maximum match distance parameter present in most variants of ICP. In addition to the demonstrated performance improvement, the proposed model allows for more expressive probabilistic models to be incorporated into the ICP framework. While maintaining the speed and simplicity of ICP, the Generalized-ICP could also allow for the addition of outlier terms, measurement noise, and other probabilistic techniques to increase robustness.**

I. INTRODUCTION

Over the last decade, range images have grown i... and found increasing applications in fiel... imaging, object modeling, and ro... and limited sensor rang... accurate metho... single m...

geometrically or with point clouds. Chen and Medioni[7] considered the more specific problem of aligning range data for object modeling. Their approach takes advantage of the tendency of most range data to be locally planar and introduces the "point-to-plane" variant of ICP. Zhang[5] almost simultaneously describes ICP, but adds a robust method of outlier rejection in the correspondence selection phase of the algorithm.

Two more modern alternatives are Iterative Dual C... dence [15] and Metric-Based ICP [16]. ... point-matching process by maint... dences. MbICP is designe... initial orientation... rotational...

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: $\arg\max_p \ldots$ / $\arg\min_p \ldots$ 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**
    $m_i \leftarrow$ `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

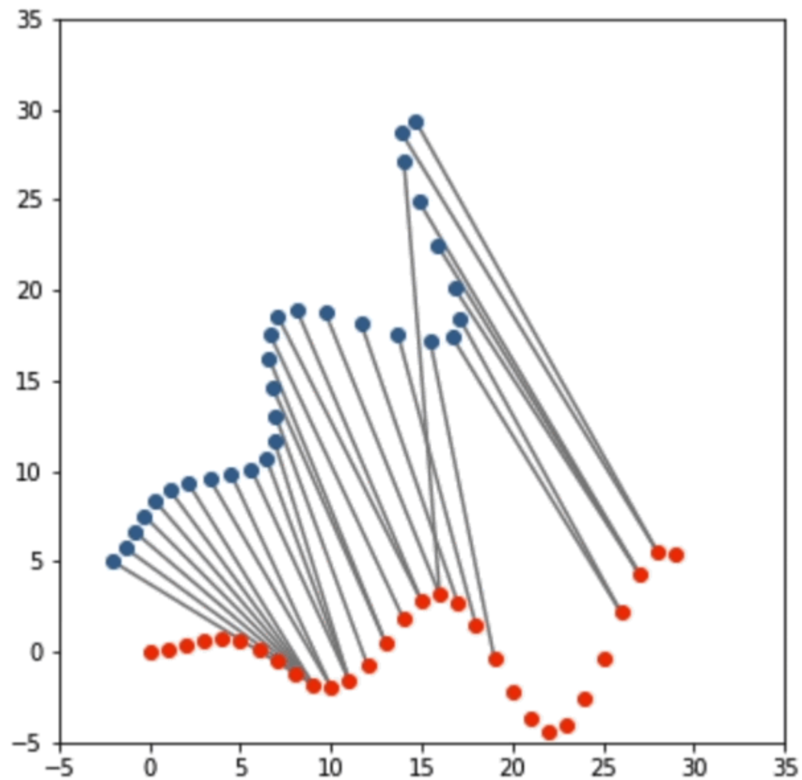


Abbildung 1: Standard-ICP (Igor Bogoslavskyi, 2021)

Theorie - Point-to-Plane-ICP

- **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

$$T \leftarrow \arg\min_T \left\{ \sum_i \left((T \cdot b_i - m_i) \cdot \boldsymbol{n_i} \right)^2 \right\}$$

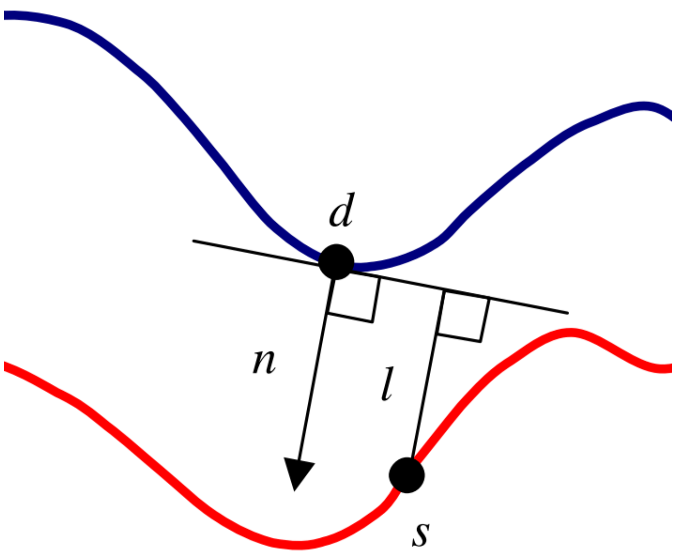


Abbildung 2: Point-to-Plane-ICP (Kok-Lim Low, 2004)

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 $\longrightarrow$ probabilistisch
  - wenn in Ebene $\longrightarrow$ Kovarianzmatrix ist „flach“

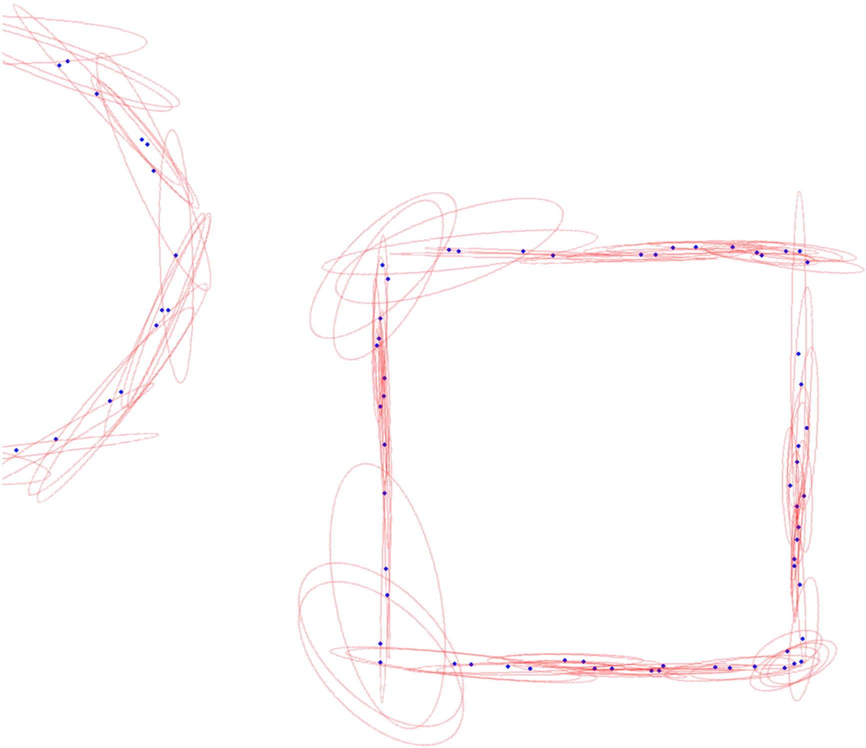


Abbildung 3: Kovarianzmatrizen (eigene Darstellung)

Theorie - GICP-Algorithmus

$T \leftarrow T_0$
while not converged **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow$ `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**
      $C_i^A \leftarrow$ `computeCovarianceMatrix`$(T \cdot b_i)$
      $C_i^B \leftarrow$ `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

$T \leftarrow T_0$
while not converged **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow$ `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**
      $C_i^A \leftarrow$ `computeCovarianceMatrix`$(T \cdot b_i)$
      $C_i^B \leftarrow$ `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

$C_i^A \leftarrow$ `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)

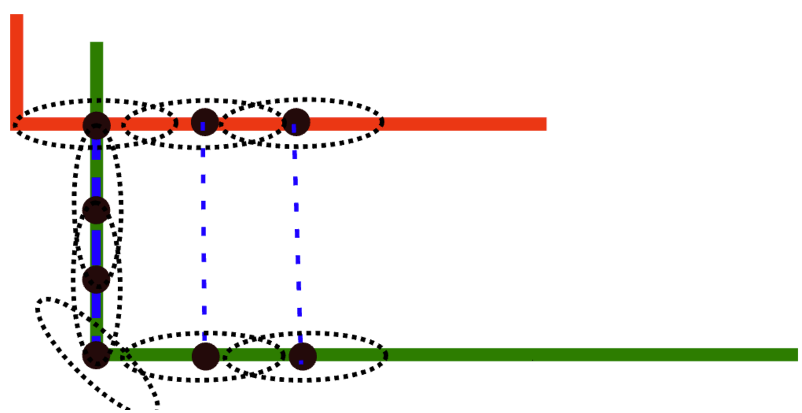


Abbildung 4: Plane-to-Plane (Segal et al., 2009)

Theorie - GICP-Algorithmus

Berechnung der Kovarianzmatrizen bei Plane-to-Plane GICP

```
covariance_ground = np.array([[epsilon, 0], [0, 1]])
covariance = np.cov(neighbors, rowvar=False)
eigenvalues, eigenvectors = np.linalg.eig(covariance)
ev = eigenvectors[:, np.argmax(eigenvalues)]     # vector with highest eigenvalue
rotation_matrix = np.array([[ev[0], -ev[1]], [ev[1], ev[0]]])
covariance_aligned = rotation_matrix @ covariance_ground @ rotation_matrix.T
```

- nicht einfach die Kovarianzmatrix der Nachbars-Punkte
- normalisiert
- Kovarianzmatrix für die Ebene erstellt: $\begin{pmatrix} \varepsilon & 0 \\ 0 & 1 \end{pmatrix}$
  - ε ist ein kleiner Wert
- Lage der Punkte im Raum betrachtet
  - Hauptkomponentenanalyse (Eigenvektoren und Eigenwerte)
  - Eigenvektor mit höchstem Eigenwert
- Einheits-Kovarianzmatrix wird um die Lage der Punkte im Raum gedreht

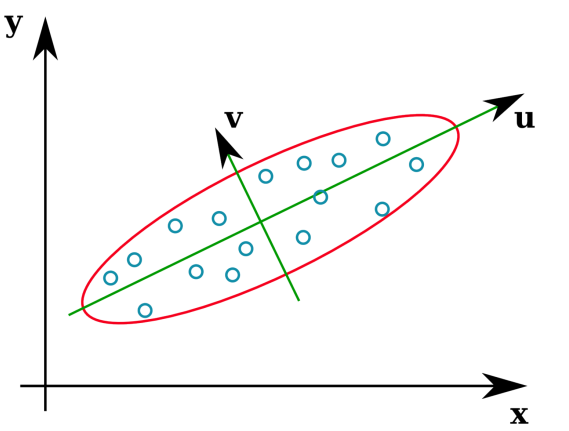


Abbildung 5: Eigenvektoren spiegeln die Lage der Punkte im Raum wider (*Intelligent Data Analysis and Probabilistic Inference Lecture 14*, 2018)

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**

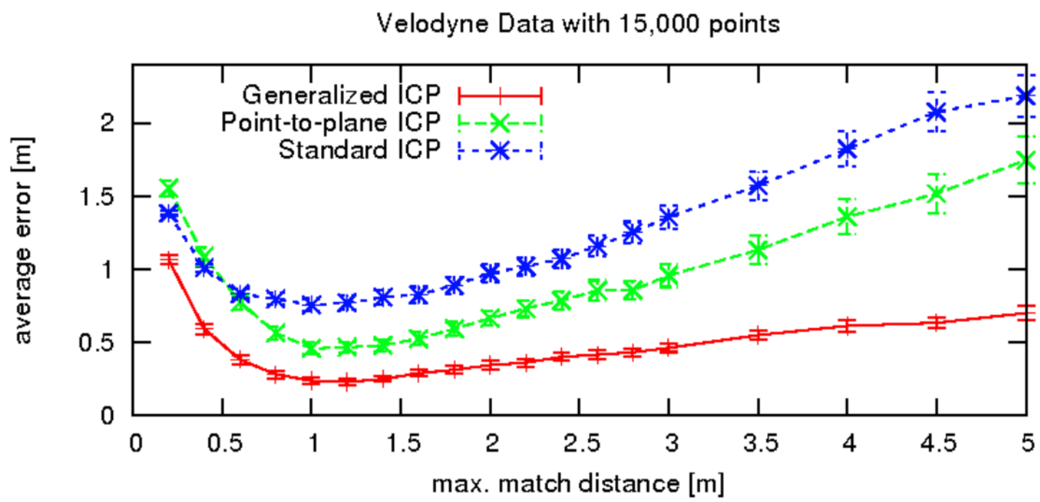


Abbildung 6: Durchschnittsfehler als Funktion von $d_{\max}$ (Segal et al., 2009)

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*

Demo: Eigene Implementierung in Python

Code Overview - GICP

GICP-Funktion:

```python
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 covariance matrices for weight matrices
        # Will be used in loss function
        ...

        # Minimize the loss function 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?

Implementierung in ROS - Versuch

Versuchsaufbau

- Vorbedingungen:
  - Bag File
  - Skript für Nodes
  - Yaml files als configuration

- Szenarien:
  - Drei unterschiedliche Maps
  - Unterschiedliche Parameterisierung
  - Fünf Durchgänge mit Standardparameterisierung (ICP & GICP)

- Auswertung:
  - Bag Files mit Topics
  - Python-Skript zur Auswertung
  - Bokeh für Visualisierung

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);
```

Implementierung in ROS - Versuch

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 - 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);
```

Implementierung in ROS - Versuch

Zeitmessung

```cpp
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);
```

Implementierung in ROS - Versuch

Problem: Aufsummierende Fehler

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

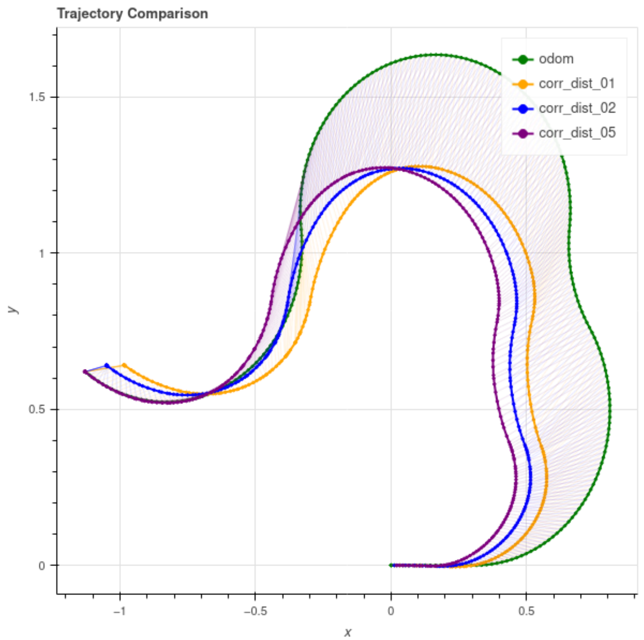


Abbildung 7: Trajectory plot with higher tick speed

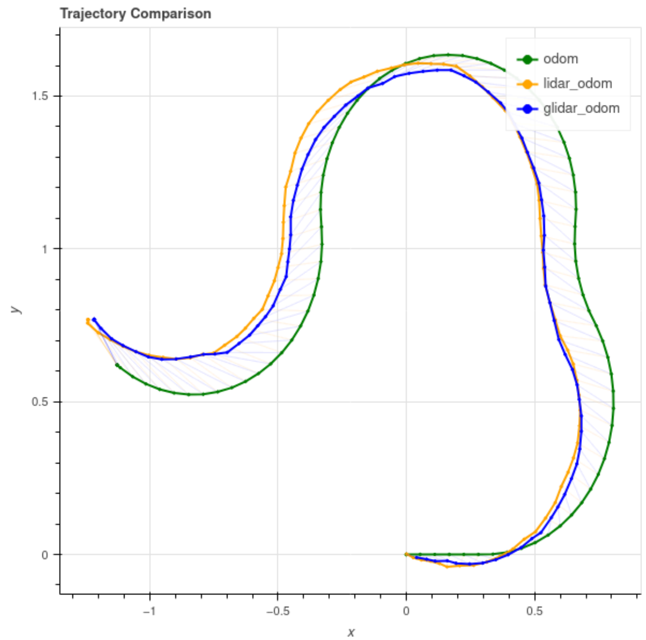


Abbildung 8: Trajectory plot with lower tick speed

Implementierung in ROS - Versuch

Turtlebot3 World

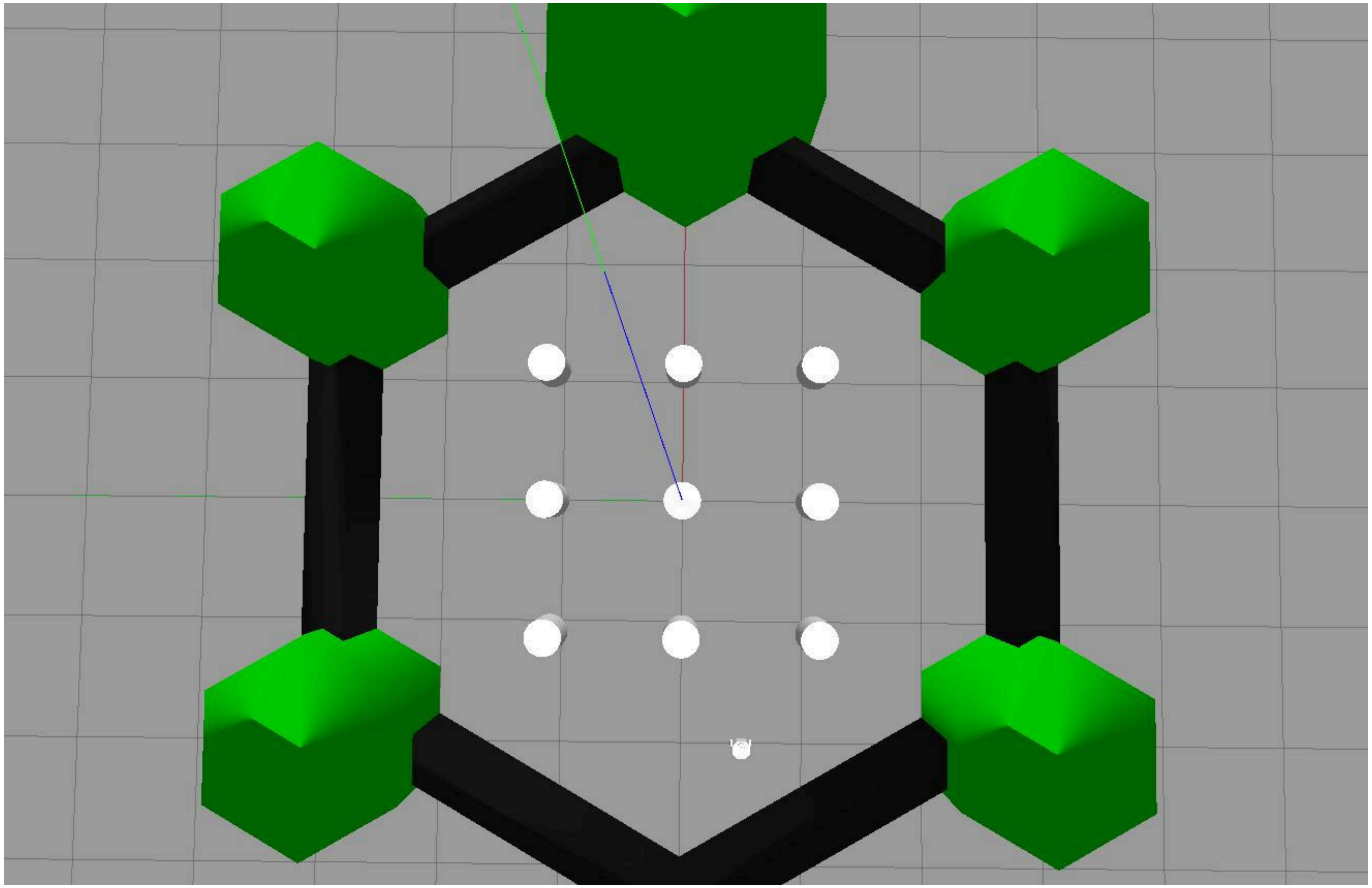


Abbildung 9: Screenshot Gazebo

Implementierung in ROS - Versuch

Turtlebot3 DQN Stage 1

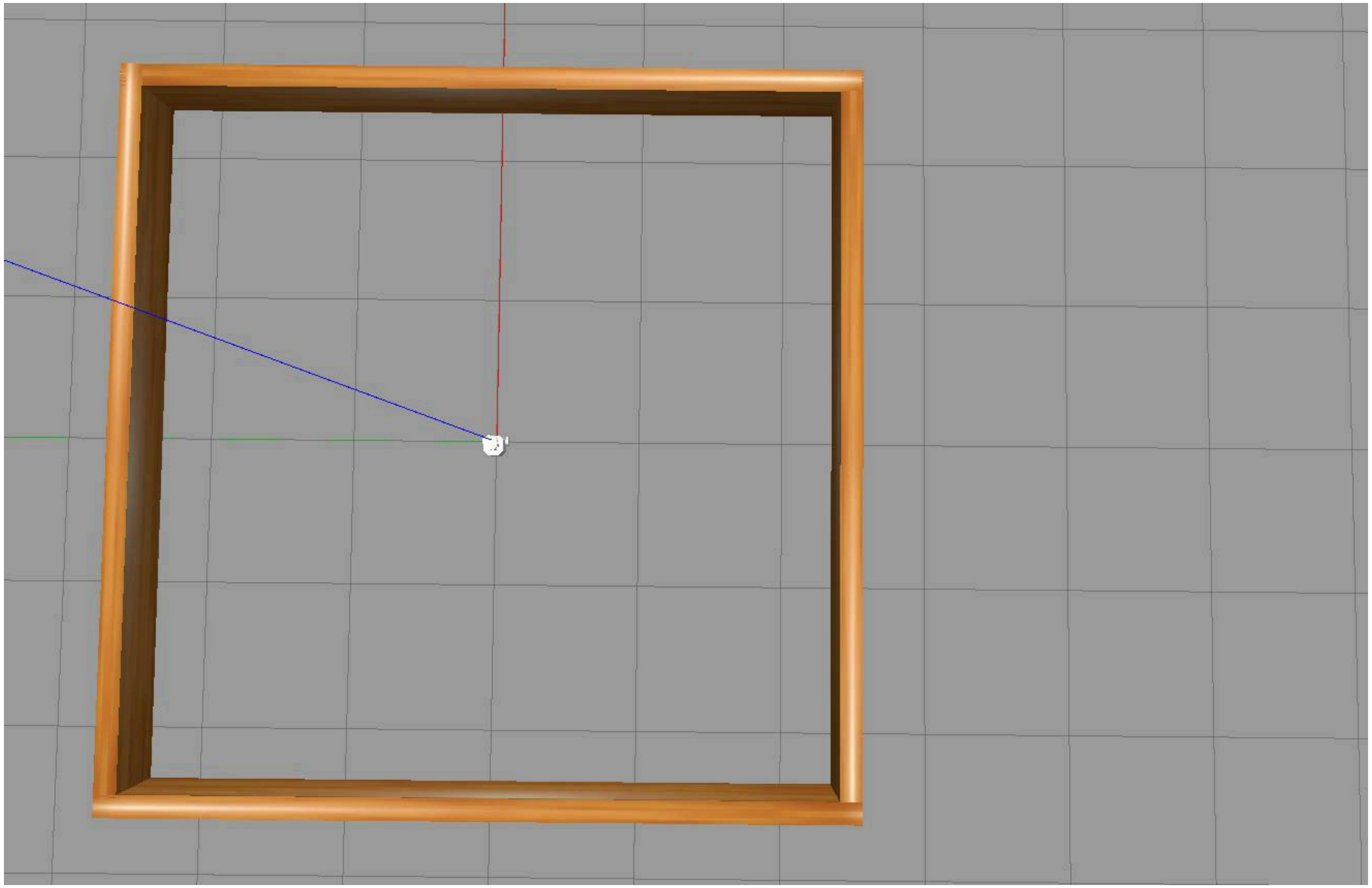


Abbildung 10: Screenshot Gazebo

Implementierung in ROS - Versuch

Turtlebot3 ICP World

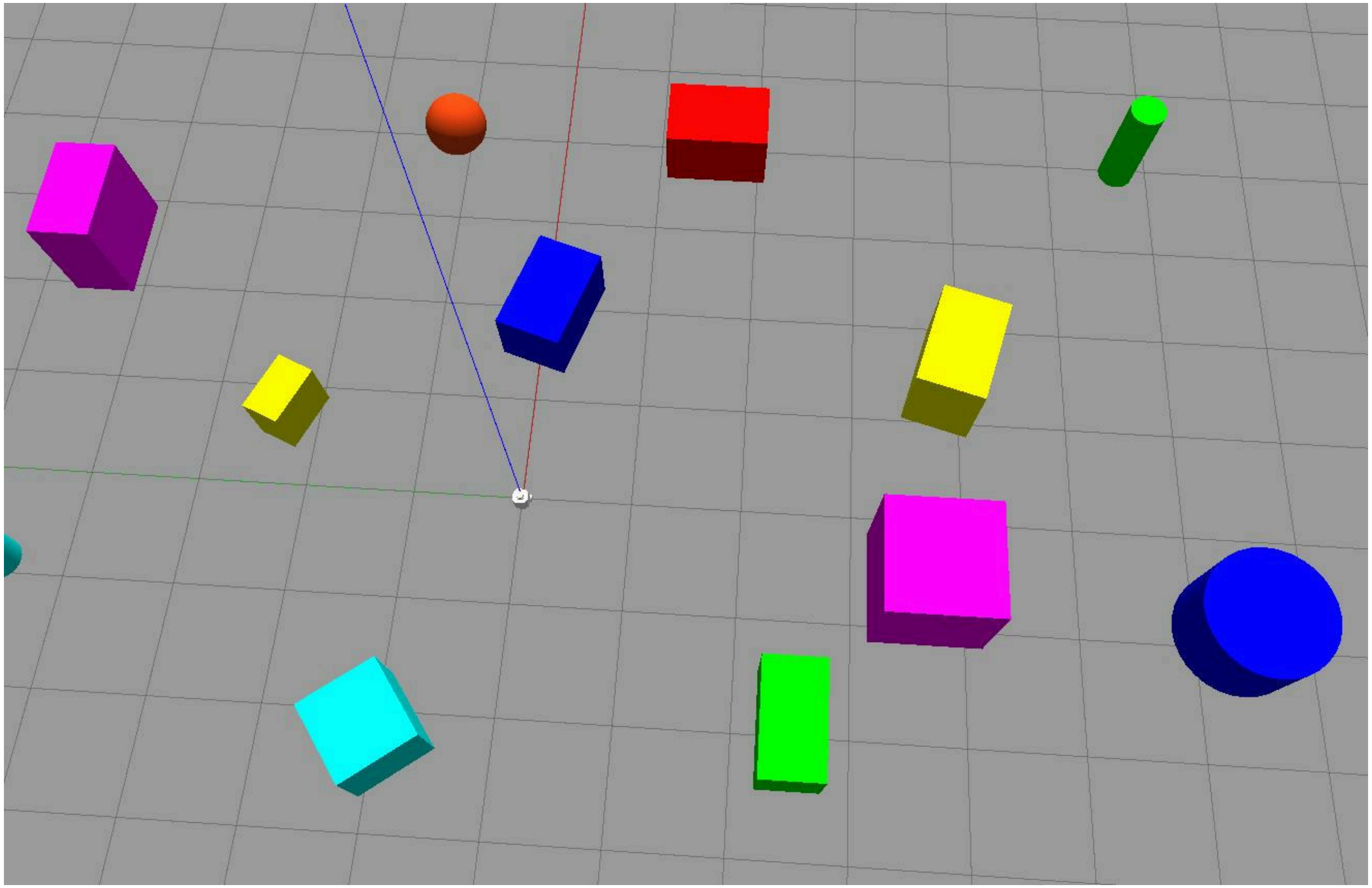
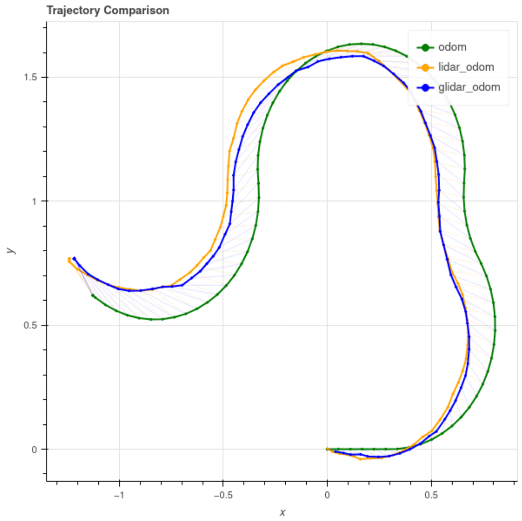
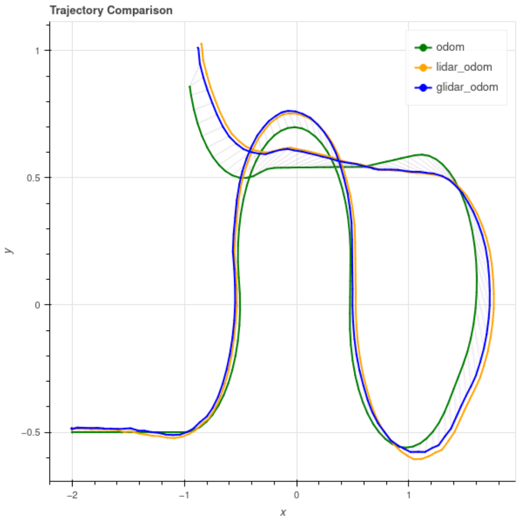


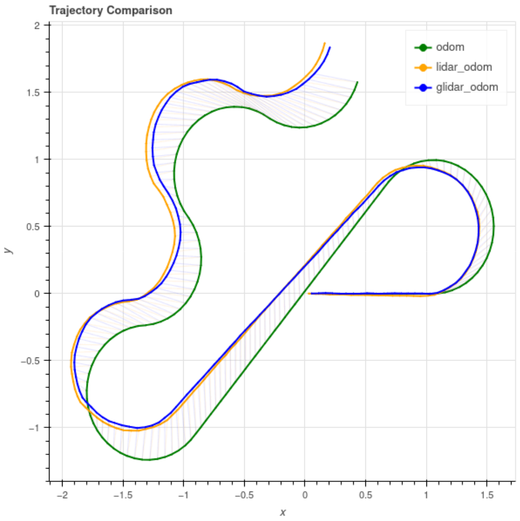
Abbildung 11: Screenshot Gazebo

Auswertung

Drei unterschiedliche Maps

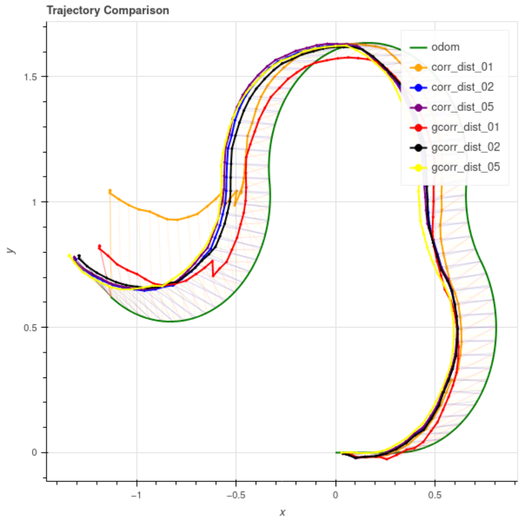


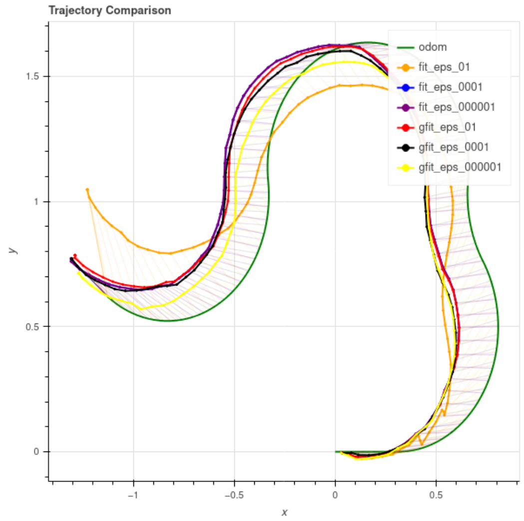


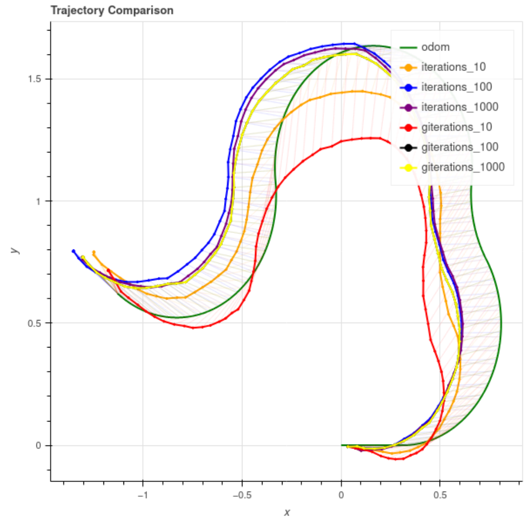


Auswertung

Unterschiedliche Parameterisierung

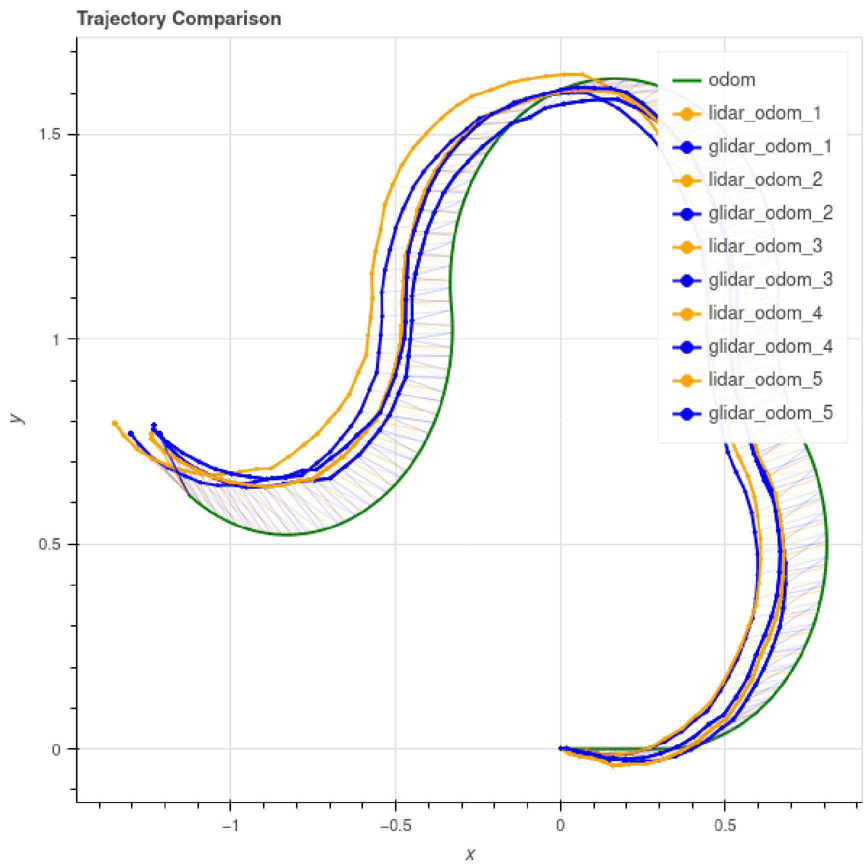






Auswertung

Fünf Durchgänge mit Standardparameterisierung (ICP & GICP)



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 großen 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

Intelligent Data Analysis and Probabilistic Inference Lecture 14. (2018). https://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/old_IDAPILecture14.pdf

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