REGISTRIERUNG VON 3D-PUNKTWOLKEN UNTER VERWENDUNG DER PCL-BIBLIOTHEK MIT SCP ALGORITHMUS

Zhu Jinyao,
Cao Bozhi

Kurzfassung: In diesem Seminarbeitrag wird der Registrierungsalgorithmus Sample Consensus Prerejective (SCP) von der PCL-Bibliothek untersucht und analysiert. Eine Testsoftware mit einem Software-Testbench und einem Punktwolkengenerator für Monte Carlo Simulation wird entwickelt. Der SCP-Algorithmus wird auf diesem Testbench ausführlich getestet und seine Leistung analysiert. Ein Vergleich mit dem ICP-Algorithmus wird ebenfalls vorgestellt.

1. EINFÜHRUNG

Punktwolkenregistrierungen werden häufig in Szenarien wie Roboternavigation, autonomes Fahren und 3D-Konstruktion verwendet. Das Problem der Ausrichtung zweier Punktwolken und der Bestimmung ihrer relativen Pose oder Transformation aus den beiden Punktwolken ist in der Computer-Vision und Robotik Gemeinde umfangreich erforscht werden. Es gibt derzeit viele Algorithmen für Punktwolkenregistrierung. Sie sind jedoch für unterschiedliche Szenarien ausgelegt und verwenden unterschiedliche Technologien. Aus der Literatur ist es oft nicht klar, wie gut sie hinsichtlich Genauigkeit, Verzögerung sowie Robustheit arbeiten, und wie die Parameter die Leistung des Algorithmus beeinflussen.

In unserer Arbeit wurde der Sample Consensus Prerejective Algorithmus (SCP) aus der PCL-Bibliothek [1] gründlich untersucht. Wir analysierten zunächst das Prinzip des SCP-Algorithmus anhand der Literatur [2] und der PCL-Bibliothek, fanden wir die Parameter heraus, die die Leistung der Registrierung beeinflussen könnten. Dann haben wir eine Testsoftware mit einem Software-Testbench für das allgemeine Registrierungsproblem entwickelt. Um genauere Ergebnisse zu erzielen, haben wir noch einen Punktwolkengenerator entwickelt, der verschiedene Quellpunktwolken erzeugen, und mit exakten Transformationen ausgeben kann.

Bei der Bewertung werden die Rotationsfehler (im Euler-Winkel), die Translationsfehler, die Güte der Schätzung und die Rechenzeit, sowie die Robustheit berücksichtigt. Am Ende dieser Arbeit haben wir auch einen Vergleich mit dem ICP-Algorithmus [3][4] durchgeführt, um eine intuitivere Ansicht von der Leistungs-fähigkeit des SCP-Algorithmus zu erhalten.

2. ALGORITHMUS

Die originale Publikation des SCP-Algorithmus erklärt den Algorithmus unmittelbar. Mithilfe des SCP-Algorithmus kann die Transformation zwischen zwei Punktwolken bestimmt werden.

2.1 Eingangs- und Ausgangsgrößen

Die Eingänge für den Algorithmus sind zwei Punktwolken mit bekannten Korrespondenzen in der Punktwolkenpaar. Der SCP-Algorithmus ist unabhängig von dem Typ der Punktwolke funktioniert. Das bedeutet, dass vor der Durchführung des SCP-Algorithmus unbedingt Features aus der Punktwolkenpaar mit Feature-Estimator berechnet werden. In dieser Arbeit wird FPFH-Feature [7] verwendet.

Als Ausgangsgrößen gibt der Algorithmus eine Transformationsmatrix, die die relative Rotation und Verschiebung zwischen den Punktwolkenpaar beschriebt, sowie ein Fitness-Score der Registrierung aus.

2.2 Definition der Rotationsmatrix

Um die relative Rotation zwischen zwei Koordinatensystem zu repräsentieren, wird in dieser Arbeit Rotationsmatrix (Direction-Cosine-Matrix) verwendet. Zwei Koordinatensystem S1 und S2 sind in Abbildung 1 dargestellt.

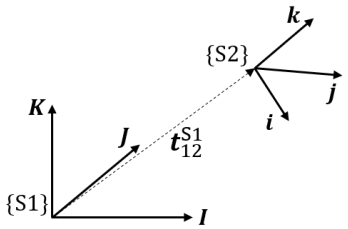

Abb.1. Referenzkoordinatensystem S1 und lokale Koordinatensystem S2

Wir nutzen folgende Notation:
Einheitsvektoren von den Achsen von S1(dargestellt in S1):

$$\boldsymbol{I}^{S1}, \boldsymbol{J}^{S1}, \boldsymbol{K}^{S1}$$

Einheitsvektoren von den Achsen von S2(dargestellt in S1):

$$\boldsymbol{i}^{S1}, \boldsymbol{j}^{S1}, \boldsymbol{k}^{S1}$$

Aus Gründen der Anschaulichkeit werden in folgenden das rechts-obere Symbol S1 vernachlässig (weil alle Vektoren werden in demselben Koordinatensystem dargestellt).
Für beliebige Einheitsvektoren $\boldsymbol{a}, \boldsymbol{b}$ gilt es:

$$\cos\big(\angle(\boldsymbol{a}, \boldsymbol{b})\big) = \boldsymbol{a} \cdot \boldsymbol{b}$$

Damit wird die DCM wie folgt definiert:

$${}^{S1}_{S2}\boldsymbol{R} = \begin{pmatrix} \boldsymbol{I}\cdot\boldsymbol{i} & \boldsymbol{I}\cdot\boldsymbol{j} & \boldsymbol{I}\cdot\boldsymbol{k} \\ \boldsymbol{J}\cdot\boldsymbol{i} & \boldsymbol{J}\cdot\boldsymbol{j} & \boldsymbol{J}\cdot\boldsymbol{k} \\ \boldsymbol{K}\cdot\boldsymbol{i} & \boldsymbol{K}\cdot\boldsymbol{j} & \boldsymbol{K}\cdot\boldsymbol{k} \end{pmatrix} \tag{1}$$

2.3 Definition der Verschiebungsvektor

Der Verschiebungsvektor wird wie folgt definiert:

$$\boldsymbol{t}_{12}^{S1} = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T \tag{2}$$

Also der Vektor, der von dem Ursprung von S1 beginnt und im Ursprung von S2 endet, sowie in S1 dargestellt.

2.4 Transformationsmatrix

Stellt man die Rotation und die Verschiebungsmatrix in eine 4x4 Matrix zusammen, dann bekommen wir die sogenannte Transformationsmatrix:

$${}^{S1}_{S2}\boldsymbol{T} = \begin{pmatrix} {}^{S1}_{S2}\boldsymbol{R} & \boldsymbol{t}_{12}^{S1} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Diese Transformationsmatrix beschreibt die relative Rotation und Translation zwischen zwei Koordinatensysteme. Damit können wir beliebigen Vektor (in homogener Koordinate), der in S2 dargestellt ist, in S1 transformieren:

$$\boldsymbol{p}^{S1} = \begin{pmatrix} I_p \\ J_p \\ K_p \\ 1 \end{pmatrix} = {}^{S1}_{S2}\boldsymbol{T} \begin{pmatrix} i_p \\ j_p \\ k_p \\ 1 \end{pmatrix} = {}^{S1}_{S2}\boldsymbol{T}\boldsymbol{p}^{S2} \tag{4}$$

2.5 Koordinatensystem

Bei der Registrierung definieren wir das Koordinatensystem der Zielpunktwolke (target) als Referenzkoordinatensystem (S1), und das Koordinatensystem der Quellpunktwolke (source) als lokale Koordinatensystem (S2) bzw. Body-Frame.

Punkte in eine Punktwolke (dargestellt im Koordinatensystem ($S \subseteq \{S1, S2\}$) der Punktwolke) sind in der homogenen Koordinate repräsentiert:

$$\boldsymbol{p}_i^{\mathrm{S}} = (x \quad y \quad z \quad 1)^T \ (i = 0,1,2,\cdots) \tag{5}$$

Die geschätzte Transformationsmatrix (Ausgang) aus dem Registrierungsalgorithmus für Punktwolken in der PCL-Bibliothek ist:

$$\prescript{\mathrm{S1}}{\mathrm{S2}}{\boldsymbol{T}} \tag{6}$$

In folgender Diskussion nutzen wir die vereinfachte Notation:

$$\boldsymbol{T} = \prescript{\mathrm{S1}}{\mathrm{S2}}{\boldsymbol{T}} \tag{7}$$

2.6 Der SCP-Algorithmus

In der Publikation wird das Registrierungsproblem von zwei Punktwolken als ein Problem der Summe der kleinsten Quadrate formuliert. Wir nutzen die Notation aus Gleichung 7, die Kostenfunktion ist (Gleichung 8):

$$\widehat{\boldsymbol{T}} = \underset{\boldsymbol{T}}{\operatorname{argmin}}\, \varepsilon(\boldsymbol{T}) = \underset{\boldsymbol{T}}{\operatorname{argmin}} \sum_{\boldsymbol{p}_i^{\mathrm{S2}} \in \mathbf{P}} \left(\boldsymbol{T}\boldsymbol{p}_i^{\mathrm{S2}} - \boldsymbol{q}_i^{\mathrm{S1}}\right)^2 \tag{8}$$

wobei $\mathbf{P}$ ist die Punktmenge der Quellpunktwolke, $\boldsymbol{p}_i^{\mathrm{S2}}$ ist der Punkt von der Quellpunktwolke, der in S2 dargestellt wird und $\boldsymbol{q}_i^{\mathrm{S1}} \in \boldsymbol{Q}$ ist die Korrespondenz von $\boldsymbol{p}_i^{\mathrm{S1}}$, der in S1 dargestellt wird.

Der SCP-Algorithmus ist basiert auf den RANSAC-Algorithmus für Punktwolkenregistrierung. Im Vergleich zum RANSAC-Algorithmus fügt der SCP-Algorithmus einen zusätzlichen Polygon-Prerejector hinzu. Der Algorithmus ist wie folgt gezeigt:

1) N(≥ 3) Punkte in der Punktmenge $\mathbf{P}$ zufällig auswählen, und ihre Korrespondenzen in der Punktmenge $\boldsymbol{Q}$ finden (mit Matching-Algorithmus z.B. Kd-tree).
2) Der gewählten Punktpaaren werden in den Polygon-Prerejector eingegeben und die Unähnlichkeit der Korrespondenzen wird geprüft. Falls die Unähnlichkeit kleiner als eine bestimmte Schwellenweit ist, zu Schritt (3) gehen, sonst zurück zu Schritt (1).
3) Die Hypothese-Transformation $\widehat{\boldsymbol{T}}$ zwischen den beiden Punktwolken mit der N Korrespondenzen schätzen.
4) Alle Punkte in $\mathbf{P}$ mit $\widehat{\boldsymbol{T}}$ auf S1 projizieren.
5) Bestimmen der Inlier-Punkte mithilfe der räumliche nächstgelegene Nachbarschaftssuche (nearest neighbor search) zwischen den transformierten Punkten aus $\mathbf{P}$ und den Punkten in $\boldsymbol{Q}$. Wenn die Anzahl der Inlier-Punkte größer als ein bestimmter Schwellenwert, gehen zu Schritt (6), sonst nach Schritt (1) zurückgehen.
6) Die Hypothese-Transformation $\widehat{\boldsymbol{T}}$ mit allen Inlier-Punktpaare erneut schätzen.
7) Die Kostenfunktion $\varepsilon(\boldsymbol{T})$ berechnen, wenn die neue Kost geringer ist, markieren $\widehat{\boldsymbol{T}}$ als die resultierende Transformation.
8) Falls die maximale Anzahl der Iterationen erreicht ist, endet sich der Algorithmus, sonst zu Schritt (1) gehen.

Beim Schritt (2) wird ein Polygon-Prerejector eingesetzt, und er ist der Kern des SCP-Algorithmus. Die Eingaben des Prerejectors sind die gewählten Korrespondenzen. In dem Polygon-Prerejector werden die Punkte, die aus den Eingaben, in jeweilige Punktmenge $\mathbf{P}$ und $\boldsymbol{Q}$ nacheinander folgend verbunden, damit zwei virtuelle Polygone (siehe Abbildung 2) konstruiert werden. Dann wird Unähnlichkeit der zwei Polygon berechnet. Die detaillierten Berechnungen sind wie folgend:

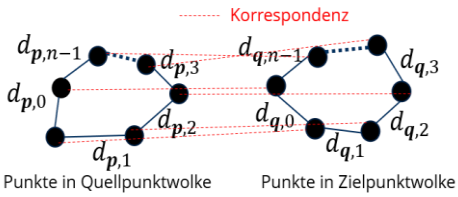

Abb.2. Virtuelle Polygone

Der Prerejector berechnet zunächst alle Kantenlängen der beiden Polygone: $d_{\boldsymbol{p},i} = \|\boldsymbol{p}_{i+1 \bmod n} - \boldsymbol{p}_i\|$ in $\mathbf{P}$, gleichfalls $d_{\boldsymbol{q},i}$ in $\boldsymbol{Q}$. Dann die Unähnlichkeit zweier Polygone wird definiert als:

$$\boldsymbol{\delta} = \left(\frac{\left|d_{\boldsymbol{p},i} - d_{\boldsymbol{q},i}\right|}{\max\left(d_{\boldsymbol{p},i}, d_{\boldsymbol{q},i}\right)}, \quad \cdots, \quad \frac{\left|d_{\boldsymbol{p},n-1} - d_{\boldsymbol{q},n-1}\right|}{\max\left(d_{\boldsymbol{p},n-1}, d_{\boldsymbol{q},n-1}\right)}\right) \tag{9}$$

Bei perfekten Korrespondenzen ist $\boldsymbol{\delta}$ ein Null-Vektor. Wenn die Unendlich-Norm von $\boldsymbol{\delta}$ größer als ein bestimmter Schwellenwert t_{poly} (siehe Gleichung 10), wird die Korrespondenzen als falsch betrachtet, und wird eine neue Iteration gestartet.

$$\|\boldsymbol{\delta}\|_\infty \leq t_{poly} \tag{10}$$

Dabei ist allerdings zu beachten, dass in der PCL-Bibliothek im Gegensatz zur Publikation die Ähnlichkeit $\boldsymbol{s}$ berechnet wird:

$$\boldsymbol{s} = \left(\frac{\min\left(d_{\boldsymbol{p},i}, d_{\boldsymbol{q},i}\right)}{\max\left(d_{\boldsymbol{p},i}, d_{\boldsymbol{q},i}\right)}, \quad \cdots, \quad \frac{\min\left(d_{\boldsymbol{p},i}, d_{\boldsymbol{q},i}\right)}{\max\left(d_{\boldsymbol{p},n-1}, d_{\boldsymbol{q},n-1}\right)}\right) \tag{11}$$

mit $\boldsymbol{s_i} \epsilon (0,1)$ und $i \in \{0,1,\cdots,n-1\}$.

Das Kriterium für den Prerejector wird:

$$\|\boldsymbol{s}\|_\infty \geq t_{poly} \tag{12}$$

2.7 Definition der Schätzfehler

Als Schätzfehler wird eine Error-Transformation definiert:

$$\begin{aligned} \boldsymbol{T} &= \widehat{\boldsymbol{T}}\boldsymbol{T}_E \\ \Rightarrow \boldsymbol{T}_E &= \widehat{\boldsymbol{T}}^{-1}\boldsymbol{T} \end{aligned} \tag{13}$$

wobei $\boldsymbol{T}$ ist die exakte Transformation oder die Referenz-transformation, $\widehat{\boldsymbol{T}}$ ist die geschätzte Transformation aus der Registrierungsalgorithmus, und $\boldsymbol{T}_E$ ist die Error-Transformation bzw. der Schätzfehler. Um die Schätzfehler anschaulicher darzustellen wird der Rotationsanteil der Error-Transformation anhand der Konvertierung, die im Vorlesungsskript [5] im Abschnitt 4.2.3 beschriebt wird, zu Yaw-Pitch-Roll Eulerwinkel konvertiert (Beachten: die Rotationsmatrix in dieser Arbeit hat unterschiedliche Definition zu der im Vorlesungsskript, daher muss die Rotationsmatrix im Vorlesungsskript zunächst transponiert werden). Die Konvertierung ist in Gleichung 14 dargestellt:

$$\begin{aligned} \theta &= \arcsin(-r_{31}) \\ \phi &= \arcsin\left(\frac{r_{32}}{\cos\theta}\right) \\ \psi &= \arcsin\left(\frac{r_{21}}{\cos\theta}\right) \end{aligned} \tag{14}$$

mit $\psi, \phi, \theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

ψ, ϕ, θ ist jeweils der Yaw, Pitch, Roll Eulerwinkel und $r_{i,j}$ ist Element aus der Rotationsmatrix. Es gibt offenbar Singularität bei der Konvertierung bei $\theta = \pm\frac{\pi}{2}$. Die Schätzfehler der Registrierung ist jedoch normalerweise nur gering ist, daher gibt es kein Problem bei der Bewertung der Leitung eines Registrierungsalgorithmus.

3. IMPLEMENTIERUNG

In diesem Absatz stellen wir ein Testbench für die allgemeinen Punktwolkenregistrierungsalgorithmen vor, die mit beliebiger Anzahl von Punktwolkenpaaren, und auch mit beliebiger Anzahl von Registrierungsalgorithmen (Testers) arbeiten können. Wir entwickeln diese Testbench, um die

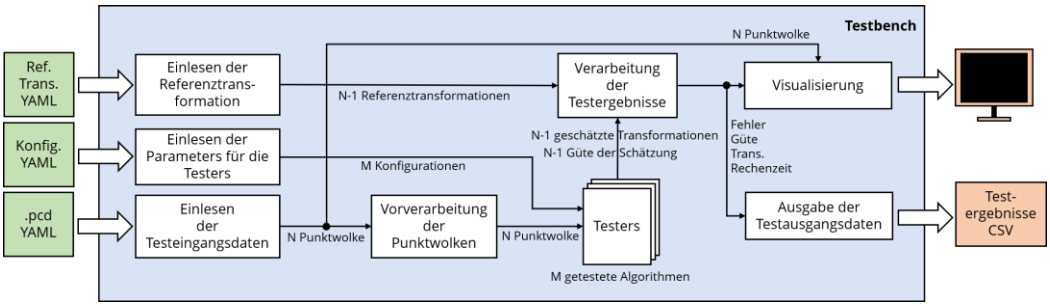

Abb.3. Struktur des Testbenchs.

Leistung des SCP-Algorithmus zu untersuchen und bewerten. Um die Robustheit des Algorithmus eingehender zu testen, haben wir auch einen Punktwolkengenerator entwickelt, der beliebige Anzahl von Quellpunktwolken mit zufällig generierten exakten Transformationen, künstliches Rauschen und Maskierungen generieren kann.

3.1 Software-Testbench

Abbildung 3 zeigt den Aufbau des Testbenchs. Das Testbench startet mit den vom Benutzer angegebenen Konfigurations-dateien. Wir verwenden Konfigurationsdateien im YAML-Format als Eingabe für das Testbench.

Die erste Eingabe ist eine YAML-Datei, die die exakten relativen Transformationen zwischen der Quellpunktwolke und der Zielpunktwolke aufzeichnet. Die zweite YAML-Datei beinhalt alle notwendigen Parameter für den Testers (den Kandidaten-algorithmen). Die dritte Eingabe ist eine YAML-Datei, die den Speicherort der PCD-Dateien im Filesystem aufzeichnet. Anhand dieser drei YAML-Dateien lädt sich das Testbench alle benötigten PCD-Dateien und Parameters automatisch auf.

Bei dem Testen nimmt das Testbench immer die erste Punktwolke als Zielpunktwolke und die alle anderen als Quellpunktwolken an, und führt dann Registrierungen von der Zielpunktwolke mit jeder Quellpunktwolke nacheinander durch.

Für alle Testers wird dieselbe Vorgang durchgeführt. Bei jeder Registrierung stellt der Tester eine geschätzte Transformation, ein Fitness-Score/Güte und eine Rechenzeit bereit.

Am Ende der Registrierungen werden die geschätzten Transformationen mit den exakten Transformationen aus der Eingabe verglichen und die Fehlertransformationen berechnet. Alle Testergebnisse werden auf dem Bildschirm dargestellt, einschließlich der geschätzten Transformationen, der Fitness-Score/Güte der Schätzung, der Rotationsfehler, der Translations-fehler, der Rechenzeiten, der Konvergenzzustände und der ausgerichteten Punktwolken. Außerdem werden auch alle Ergebnisse in Bezug auf die Leistung des Algorithmus, die Fitness-Score/Güte der Schätzung, die Rotationsfehler, die Translationsfehler, die Rechenzeiten, die Konvergenzzustände und die geschätzten Transformationen, in eine CSV-Datei exportiert.

3.2 Punktwolkengenerator mit Referenztransformationen

Abbildung 4 zeigt die Struktur des Punktwolkengenerators. Als Eingabe wird eine PCD-Datei von dem Benutzer gegeben. Nach der Ladung der PCD-Datei wird die Punktwolke vorverarbeitet. Im Vorverarbeitungsschritt wird der Schwer-punkt der Eingangspunktwolke berechnet. Position jedes Punktes in der Punktwolke werden dann um die Position dieses Schwerpunktes subtrahiert, dadurch der Schwerpunkt am Ursprung des Koordinatensystems der Punktwolke platziert wird, der resultierte Punktwolke wird als Zielpunktwolke dient. Das Ziel dieser Vorverarbeitung besteht darin, dass nach dieser Vorbereitung die Punktwolke direkt um ihren Schwerpunkt mit eine Rotationmatrix gedreht werden kann, ohne die Position der Schwerpunkt zu ändern. Was noch wichtiger ist, dass diese Manipulation eine einfache Weise bietet, Teil der Punktwolke zu maskieren, die in dem nächsten Absatz vorgestellt wird.

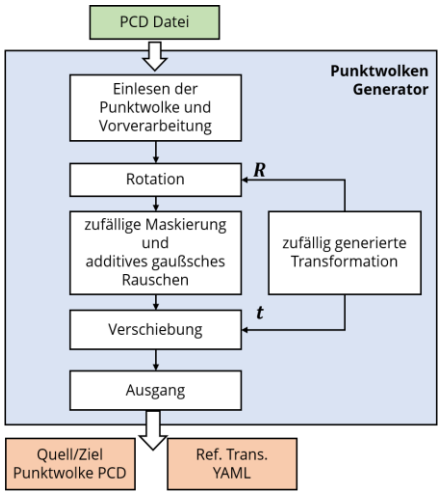

Abb.4. Struktur des Punktwolkengenerator

Nach der Vorverarbeitung wird eine Transformationsmatrix zufällig generiert, mit vorgegebenen maximalen Drehwinkeln (in Yaw-Pitch-Roll Eulerwinkel) und maximalen Translationen. Dann wird nur der lineare Anteil der Transformationsmatrix bzw. die Rotationsmatrix verwendet, um die Punktwolke um ihren Schwerpunkt zu drehen.

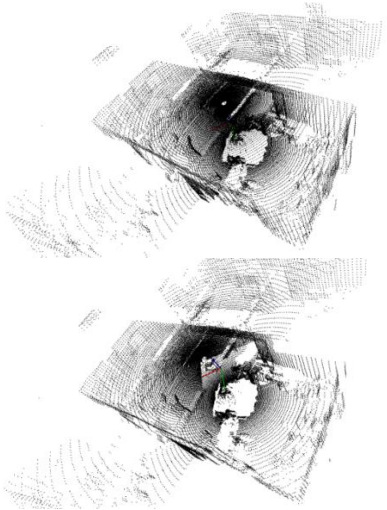

Abb. 5. Zufällige Maskierung der Punktwolken

Nach der Drehung wird eine Ganzzahl aus eine Gleich-verteilung im Bereich von 1 bis 8 abgetastet, die den 8 Quadranten des 3D-kartesischen Koordinationssystems der gedrehten Punktwolke entspricht. Gemäß dieser Ganzzahl

werden alle Punkte im entsprechenden Quadranten maskiert bzw. entfernt (vgl. Abbildung 5). Für alle anderen Punkte wird jeder von ihnen mittelwertfreies Gauß'sches weißes Rauschen, die mit vorgegebener Standardabweichung generiert werden, hinzugeführt, in allen drei Richtungen.

Anschließend wird die Punktwolke mit dem translatorischen Anteil der Transformationsmatrix verschoben. Die resultierende Punktwolke wird als Quellpunktwolke dient. Am Ende des Programms wird die Quell-/Zielpunktwolke in separaten PCD-Dateien abgespeichert. Die exakte Transformation wird in einer YAML-Datei geschrieben, die von dem Testbench gelesen werden kann.

3.3 Versuchsaufbau für den SCP-Algorithmus

In dem Versuch nutzen wir PCL-Bibliothek in der Version-1.8.1. Punktwolke in Typ *pcl::PointXYZ* von der PCL-Bibliothek wird verwendet. Aber es ist nötig zu erwähnen, dass der SCP-Algorithmus unabhängig von dem Datentyp ist. Da er basiert auf bekannte Korrespondenzen aus der beiden Punktwolken. Vor der Ausführung des SCP-Algorithmus müssen unbedingt Features mit anderem Algorithmus aus der beiden Punktwolken berechnet werden (wie das in Abschnitt 2.1 schon genannt). In der Regel kann die Qualität der Features den Schätzfehler des SCP-Algorithmus beeinflussen.

Ähnlich wie das Einführungsbeispiel [6] von PCL-Bibliothek verwenden wir FPFH-Deskriptor. Um FPFH-Deskriptor aus der Punktwolke zu berechnen, brauchen wir Oberflächennormal (surface normal) aus der Punktwolke. Wir berechnen die Oberflächennormal mit der Klasse *pcl::NormalEstimationOMP* [8] von der PCL-Bibliothek. Die Klasse *pcl::NormalEstimationOMP* hat dieselbe API wie die Klasse *pcl::NormalEstimation*, außer dass sie Multithread-Paradigmen von OpenMP [9] verwendet, um die Berechnung zu beschleunigen. Bei der Berechnung des FPFH-Deskriptors wird die Klasse *pcl::FPFHEstimationOMP* [10] verwendet, also die Multithread Version von *pcl::FPFHEstimation*. Vor der Berechnung des Oberflächennormals und FPFH-Deskriptor muss man den Search Radius einstellen, mit der Funktion *setRadiusSearch()*. Dieser Search-Radius ist der Kugelradius, der als maximale Entfernung zur Bestimmung eines Nachbarn verwendet wird.

Der SCP-Algorithmus ist in der Klasse *pcl::SampleConsensusPrerejective* [1] definiert. Für die Registrierung mit dem SCP-Algorithmus können 6 Parameter adjustiert werden:

- *number of samples*: die Anzahl (≥ 3) der gewählten Punktpaaren (Korrespondenzen) für die Schätzung einer Transformation bei jeder Iteration.
- *correspondence randomness*: die Anzahl der Nachbarn, die bei der Auswahl einer zufälligen Korrespondenz verwendet.
- *similarity threshold*: der Schwellenwert $\in [0,1)$ für die Kantenlängenähnlichkeit des Polygons.
- *max. iterations*: die maximalen Iterationen des RANSAC-Algorithmus.
- *max. correspondence distance*: die maximale Entfernungs-schwellenwert zur Bestimmung der Inlier-Korrespondenz zwischen der Quell-/Zielpunktwolke.
- *inlier fraction*: die erforderliche minimale Inlier-Fraktion zur Schätzung einer Transformation.

Als Eingangspunktwolken verwenden wir die PCD-Dateien *room_scan1.pcd* und *room_scan2.pcd* aus der PCL-Bibliothek [11]. Es handelt sich um die Punktwolken aus einem 3D-Laserscanner in einem großen Raum, in Typ von *pcl::PointXYZ*. Jede PCD-Datei beinhalt ungefähr 100,000 Punkten. Deshalb ist es sinnvoll, dass Down-Sampling in der Vorverarbeitung Schritt in dem Testbench durchzuführen. Wir setzten die leaf-size des Grid-Filter auf 0.5 Meter. Nach dem Down-Sampling wird die Anzahl der Punkte auf etwa 1300 reduziert, welche die Anzahl der Deskriptoren (1255 und 1448) in der originalen Publikation des SCP-Algorithmus approximiert.

Es scheint, dass es keine Referenztransformation zwischen diesen Punktwolken gibt. Deshalb wollten wir diese Transformation selbst finden. Um die exakte Transformation zu bestimmen, registrierten wir die beide Punktwolken mit dem ICP-Algorithmus aus PCL-Bibliothek [12]. Mit einer hinreichend großen Iterationsanzahl (z.B. 10000) sowie einer hinreichend kleine Transformations-epsilon (z.B. $1 \times 10^{-10}\mathrm{m}^2$, maximal zulässige quadratische translatorische Differenz zwischen zwei aufeinander folgenden Transformationen) können wir eine sehr exakte Transformation (siehe Gleichung 15) erhalten.

$$\boldsymbol{T} = \begin{pmatrix} 0.7569 & 0.6534 & -0.0174 & -1.6213 \\ -0.6534 & 0.7570 & 0.0013 & 1.3103 \\ 0.0140 & 0.0104 & 0.9998 & -0.0518 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (15)$$

Diese Transformation wird im folgenden Versuch als Referenztransformation dient. Die registrierte Punktwolken sind in Abbildung 6 dargestellt.

4. TESTERGEBNISSE

Wir stellen hier ausführliche Testergebnisse von dem SPC-Algorithmus vor. Weil wir ein leaf-size von 0.5 Meter für Down-Sampling nutzen, brauchen wir einen hinreichend großen Search-Radius für dem Feature-Deskriptor. Hier nutzten wir Search-Radius von 2.0m für das Oberflächennormal (surface normal) und 2.5m für den FPFH-Deskriptor. Die Parameter für den SCP-Algorithmus werden in Tabelle 1 dargestellt.

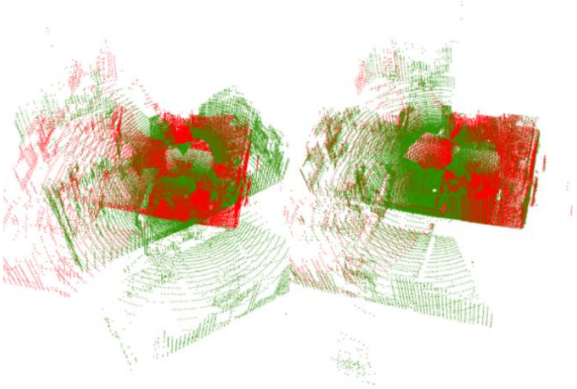

Abb.6. Punktwolkenpaar für den Versuch (links), und die registrierten Punktwolken (rechts).

search radius, surface normal	2.0m
search radius, FPFH	2.5m
number of samples	3
correspondence randomness	5
similarity threshold	0.9
max. iterations	5000
max. correspondence distance	1.5m
inlier fraction	0.3

Tab.1. Parameter für den SCP-Algorithmus beim Versuch

Der Versuch wurde auf einem Notebook mit Microsoft Windows Betriebssystem durchgeführt, die detaillierten Konfigurationen ist in Tabelle 2 angezeigt.

CPU	Intel Core i7-6560U @2.20GHz
RAM	8GB @1867MHz
OS	Windows 10 Home 64bit
Mainboard	Dell XPS-9350

Tab.2. Konfiguration der Testplattform

Da beim SPC-Algorithmus handelt es um einem Stichprobenprozess bei der Auswahl der Korrespondenzen, könnten wir bei jedem Versuch unterschiedliche Ergebnisse erhalten. Deswegen führten wir 30 Registrierungen für die Punktwolken *room_scan1.pcd* und *room_scan2.pcd* durch, und dann verglichen die geschätzten Transformationen mit der Referenztransformation.

4.1 Güte der Schätzungen

Abbildung 7 zeigt die Güte der Schätzungen.

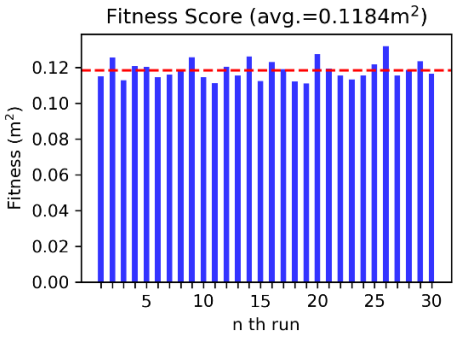

Abb.7. Güte der Schätzungen/Fitness-Score, mit einem Mittelwert von $0.1184m^2$

Diese Güte stammt aus den Fitness-Score von jeder Registrierung. In der PCL-Bibliothek der Fitness-Score wird definiert als: die mittlere Distanz-Fehlerquadrate (MES) der Korrespondenzen der Inliers. Es ist zu erwähnen, dass diese Anzahl von dem Szenario und der Punktwolkenpaar abhängt ist und nur in demselben Szenario verglichen werden soll. Sie kann normalerweise als ein Qualitätsmerkmal für dieselbe Registrierung dienen. Die Ergebnisse zeigt einen relativ stabilen Verlauf der Fitness-Score (siehe Abbildung 6) beim Versuch, mit einem Mittelwert von $0.1184\mathrm{m}^2$.

4.2 Rotationsfehler

Abbildung 8 zeigt die absoluten Rotationsfehler in Yaw-Pitch-Roll Eulerwinkeln bei der 30 Registrierungen. Aus den Ergebnissen ist es ersichtlich, dass der maximale absolute Rotationsfehler etwa 2.4 Grad beträgt. Die Abbildung 9 zeigt die Statistik dieser Rotationsfehler.

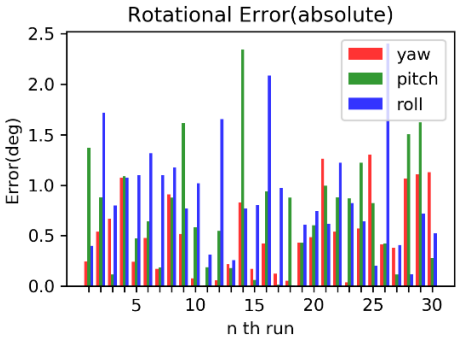

Abb.8. Absolute Rotationsfehler, mit einem Maximum von circa 2.4 Grad in Pitch-Winkel

	Yaw (Grad)	Pitch (Grad)	Roll (Grad)
μ	-0.0701	0.0712	-0.4339
δ	0.6441	0.9304	0.9398

Tab.3. Mittelwerte und Standardabweichungen der Rotationsfehler

Der Mittelwerte und Standardabweichungen sind in Tabelle 3 dargestellt. Die Rotationsfehler in den drei Richtungen weisen eine ähnliche Verteilung auf. Die durchschnittlichen Abweichungen liegen unter 0,5 Grad, die Standard-abweichungen sind jedoch relativ groß, um circa 1.0 Grad.

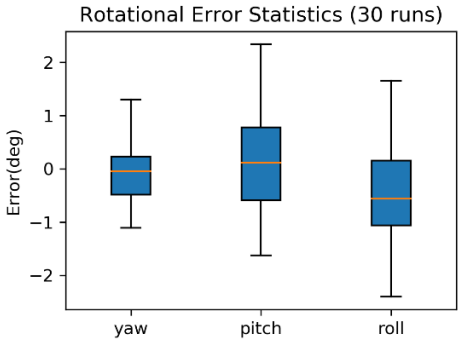

Abb.9. Rotationsfehler Statistik, Whisker = 1.5 × IQR

4.3 Translationsfehler

Abbildung 10 zeigt die absoluten Translationsfehler in X/Y/Z-Richtungen aus der 30 Registrierungen. Der maximale absolute Translationsfehler beträgt circa 0.25m.

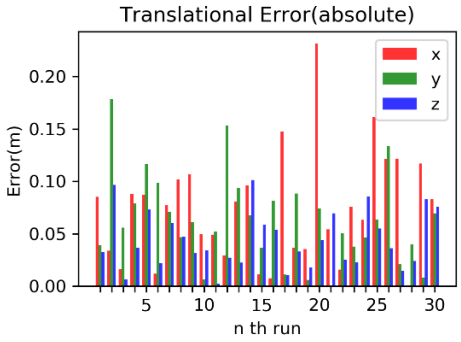

Abb.10. Absolute Translationsfehler, mit einem Maximum von circa 0.25m in X-Richtung.

Abbildung 11 zeigt die Statistik von der Translationsfehler. Die Translationsfehler in der X/Y-Richtung weisen eine ähnliche Verteilung auf, und in Z-Richtung wird eine relativ geringere Abweichung dargestellt. Der Grund dafür könnte sein, dass es sich um eine ebene Bewegung des Sensors handelt und nur geringe Verschiebung in Z-Richtung gibt (vgl. Gleichung 15).

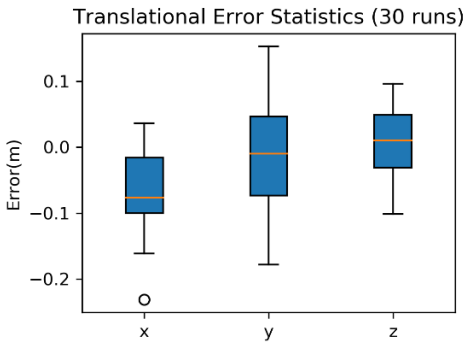

Abb.11. Translationsfehler Statistik, es gibt in Z-Richtung eine relativ geringere Abweichung.

Tabelle 4 zeigt die Mittelwerte sowie die Standard-abweichungen der Translationsfehler in X/Y/Z-Richtung.

	X (m)	Y (m)	Z (m)
μ	-0.0652	-0.0160	0.0063
δ	0.0613	0.0742	0.0506

Tab.4. Mittelwerte und Standardabweichungen der Translationsfehler

4.4 Rechenzeit

Abbildung 12 zeigt die Rechenzeit von den 30 Registrierungen.

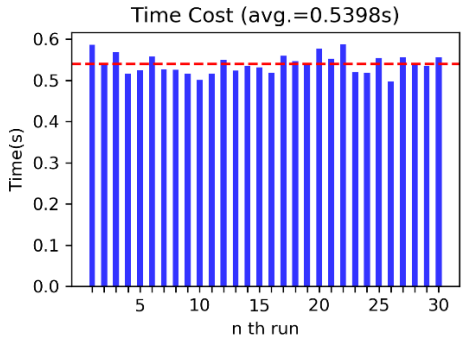

Abb.12. Rechenzeit

In dem Versuch werden 1754 (Zielpunktwolke) und 1339 (Quellpunktwolke) Deskriptoren aus der Punktwolkenpaar berechnet und als Eingabedaten des SCP-Algorithmus gegeben. Bei jeder Registrierung beträgt die Rechenzeit der Feature-Extraktion um circa 80 Millisekunden, welche deutlich geringer als die Rechenzeit des SPC-Algorithmus.

Die in Abbildung 12 dargestellten Rechenzeiten umfassen die Verarbeitungszeit der Feature-Extraktion. Aus den Ergebnissen beträgt die durchschnittliche Rechenzeit der Registrierung um 0.5398 Sekunden. Wenn wir die *similarity threshold* auf null setzten, d.h. der SCP-Algorithmus als ein normaler RANSCA-Algorithmus zu nutzten, beträgt die Rechenzeit um etwa 8.0 Sekunden, also 14.82 Fach größer als die vom SPC-Algorithmus, welches dem Ergebnis (15 Fach) in der originalen Publikation nahe ist. Abbildung 13 zeigt die Verteilung der Rechenzeiten.

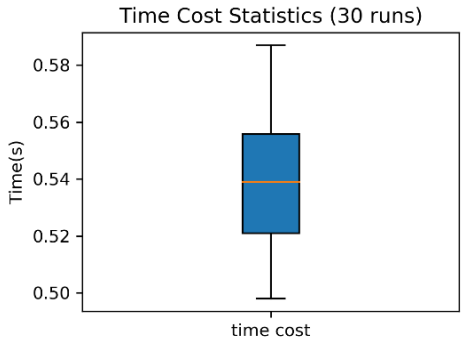

Abb.13. Rechenzeit Statistik

4.5 Robustheit gegen Parameteränderungen

Hier untersuchen wir, wie die Parameter die Leistung des SCP-Algorithmus beeinflussen.

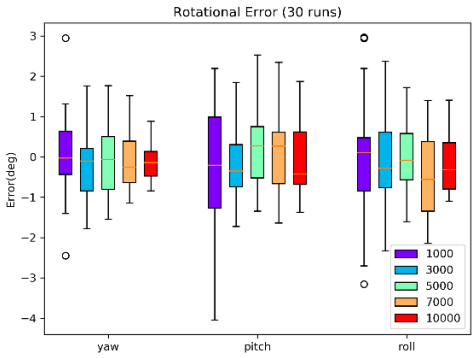

Abb.14. Rotationsfehler bei der Änderung des Parameters *max. iterations*

Max. Iterationen (*max. iterations*)

Abbildung 14 zeigt die Abhängigkeit der Rotationsfehler von der maximalen Anzahl der Iterationen. Aus den Testergebnissen können wir sehen, dass bei größerer Iterationsanzahl reduziert sich der Rotationsfehler, insbesondere der maximale Rotationsfehler. Wir können sehen, dass je größer die Anzahl der Iterationen ist, desto geringer sind die Rotationsfehler sowie die Standardabweichung der Fehler. Dies ist normalerweise für den Algorithmus mit Iterationsvorschrift gültig.

Abbildung 15 zeigt die Translationsfehler. Die Translationsfehler verhalten sich auch ähnlich wie die Rotationsfehler bei der Änderung der maximalen Anzahl der Iterationen.

Die Änderung der Rechenzeiten ist in Abbildung 16 dargestellt. Wir können sehen, dass, obwohl es geringere Schätzfehler bei großen Iterationsanzahl gibt, steigert sich die Rechenzeit und die ihre Unsicherheit.

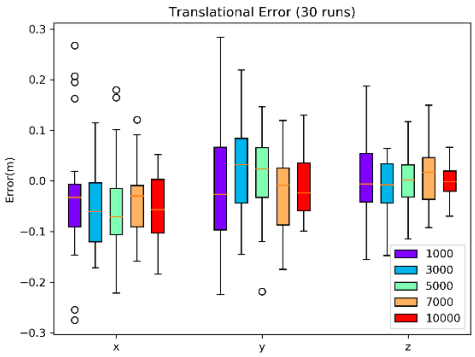

Abb.15. Translationsfehler bei der Änderung des Parameters *max. iterations*

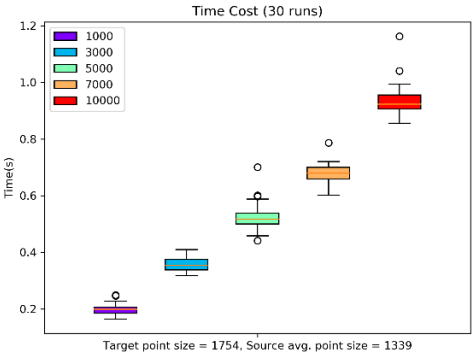

Abb.16. Rechenzeiten bei der Änderung des Parameters *max. iterations*

Schwellenwert der Ähnlichkeit (*similarity threshold*)

Abbildung 17, 18, 19 zeigen, dass wie das Parameter *similarity threshold* die Leistung der Registrierung beeinflusst.

Aus Abbildung 17 können wir sehen, dass *similarity threshold* keinen deutlichen Einfluss auf die Rotationsfehler hat, und auch auf die Translationsfehler (siehe Abbildung 18).

Nach dem Prinzip des SPC-Algorithmus, wenn wir größeren Schwellwert von Ähnlichkeit setzen, werden mehr Punktpaare, die möglich mit falsch Korrespondenz, beim Prerejection-Schritt abgelehnt, und wird die Rechenzeit dadurch reduziert. Dies stimmt mit den Ergebnissen aus Abbildung 19 überein. Falls der Schwellwert auf 0 gesetzt wird, wird der SCP-Algorithmus zu einem normalen RANSAC-Algorithmus für Punktwolken-registrierung.

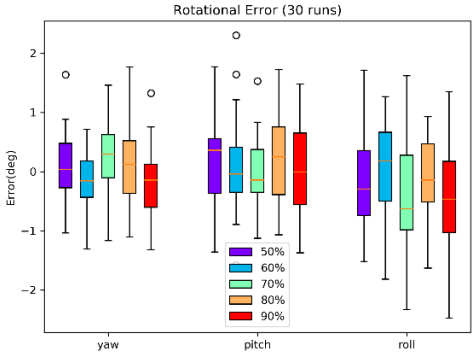

Abb.17. Rotationsfehler bei der Änderung des Parameters *similarity threshold*

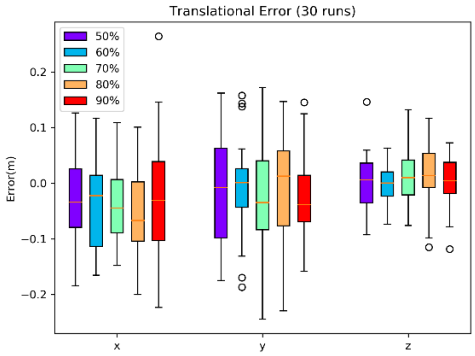

Abb.18. Translationsfehler bei der Änderung des Parameters *similarity threshold*

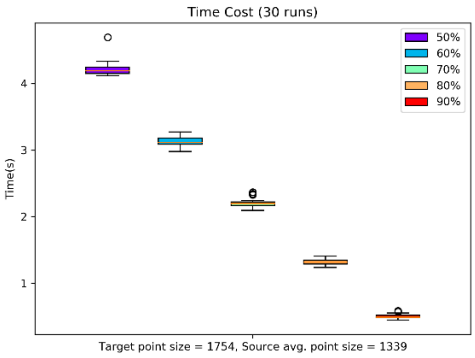

Abb.19. Rechenzeiten bei der Änderung des Parameters *similarity threshold*

Correspondence Randomness

Abbildung 20, 21, 22 zeigen die Leistung des Algorithmus in Abhängigkeit von dem Parameter *correspondence randomness*.

Aus der Rotationsfehler (siehe Abbildung 20) und der Translationsfehler (siehe Abbildung 21) können wir sehen, dass der Algorithmus bei kleinerer *correspondence randomness* relativ geringere Abschätzungsfehler hat. Aber dies nur in dem Fall gütig, wenn wir hochwertige Feature Korrespondenzen haben. Das Ziele des Parameter *correspondence randomness* besteht darin, die Robustheit des Algorithmus gegen unzuverlässige Korrespondenzen zu erhöhen. Die Kosten sind jedoch der Verlust der Genauigkeit.

Abbildung 22 zeigt die Rechenzeit in Abhängigkeit von *correspondence randomness*. Wir können sehen, dass, je größer die Anzahl der *correspondence randomness* ist, desto geringer ist die Rechenzeit. Der Grund dafür ist, dass bei der Auswahl einer zufälligen Korrespondenz werden mehrere Nachbarn verwendet, und könnten mehr Korrespondenzkombination auftreten, trotz viele von ihnen falsch gewählt sind. Mehr Punktpaare werden daher von dem Prerejector abgelehnt, sowie geringere Punktpaare werden bei der Registrierung verwendet, und führt zu eine geringere Rechenzeit, auch eine geringere Genauigkeit. Deshalb bei großer *correspondence randomness* ist es günstig, ein große Iterationsanzahl zu wählen.

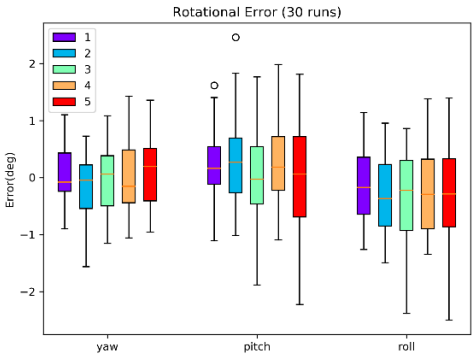

Abb.20. Rotationsfehler bei der Änderung des Parameters *correspondence randomness*

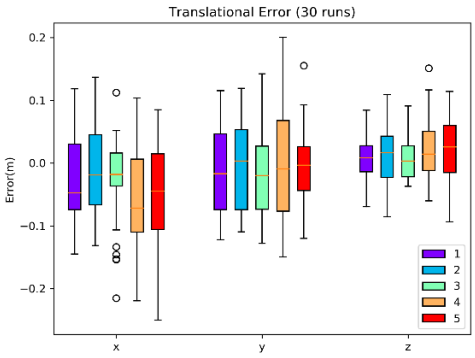

Abb.21. Translationsfehler bei der Änderung des Parameters *correspondence randomness*

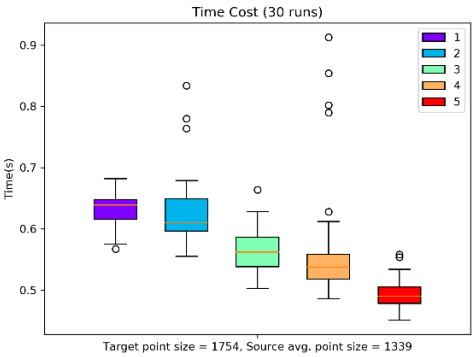

Abb.22. Rechenzeiten bei der Änderung des Parameters *correspondence randomness*

Für das Parameter *number of samples* haben wir gefunden, dass bei großer Anzahl sich die Schätzungsfehler vergrößert, und sich die Rechenzeit verkleinert. Bei zu großer Anzahl gibt es mögliches Konvergenzproblem bei der Registrierung. Der Grund dafür könnte sein, bei großer Anzahl von *number of samples* die Bedingung für die Ähnlichkeit des Polygons für die Prerejector strenger wird, d.h. mehr Korrespondenzen abgelehnt wird. Deshalb führt zu geringere Rechenzeit und geringere Genauigkeit der Registrierung. Zu einem anderen Aspekt beeinflusst diese Anzahl die Erfolgsrate des RANSCA-Algorithmus, daher bei großer Anzahl Konvergenzproblem gibt.

Die anderen Parameter, *max. correspondence distance* und *inlier fraction* beeinflusst die Leistung der Registrierung fast nicht.

4.6 Monte Carlo Simulation

Um die Leistung des SPC Algorithmus umfangreicher zu auswerten, machten wir Monte Carlo Simulation unter Nutzung des Punktwolkengenerators, der in Sektion 3.2 vorgestellt wurde, und verwendeten wir die exakten Transformationen aus dem Generator.

In dem Versuch werden 100 Quellpunktwolken gengeriert, mit zufälligem Rauschen und zufällige Maskierungen. Als Zielpunktwolke wurde die PCD Datei *room_scan1.pcd* verwendet. Die Punktanzahl der Zielpunktwolke beträgt 1294 und die durchschnittliche Punktanzahl der Quellpunktwolken beträgt 1278.

Die Parameter für den SCP-Algorithmus sind mit denen in der Tabelle 1 identisch. Für den Punktwolkengenerator setzten wir die maximalen Verschiebungen in X/Y/Z-Richtung auf ± 1.0 Meter, den maximalen Yaw-Winkel auf ± 180 Grad, den maximale Pitch-/Roll-Winkel auf ± 30 Grad und die Standard-abweichung des Rauschens auf 0.05 Meter in X/Y/Z-Richtung.

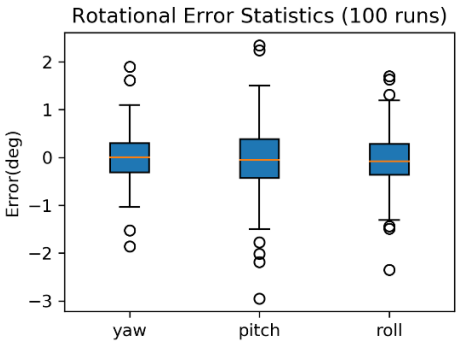

Abb.23. Rotationsfehler Statistik aus 100 Simulationen

Abbildung 23 zeigt die Verteilung der Rotationsfehler aus der 100 Versuchen. Die Verteilung ist ähnlich wie die in Abbildung 9. Die Verteilung der Translationsfehler sind in Abbildung 24 dargestellt. Also die Verteilung ähnlich wie die in Abbildung 11. Der SCP-Algorithmus zeigt eine Robustheit bezüglich seiner Genauigkeit gegen zufälliges Rauschen und zufällige Maskierungen.

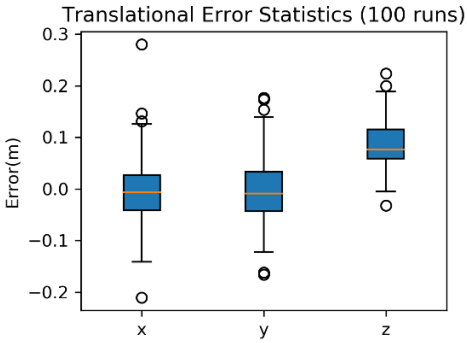

Abb.24. Translationsfehler Statistik aus 100 Simulationen

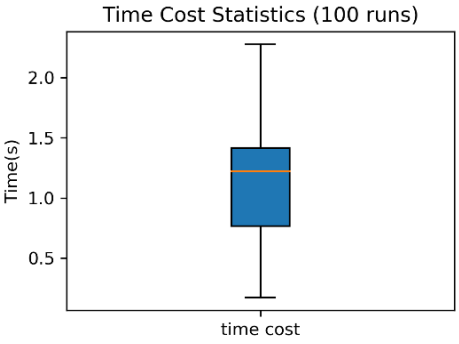

Abb.25. Rechenzeit Statistik aus 100 Simulationen

Abbildung 25 zeigt die Rechenzeit Statistik des SCP-Algorithmus im Versuchen. Im Gegensatz zu der Abbildung 13 ist die Standardabweichung der Rechenzeit hier relativ groß. D.h. die erforderliche Rechenzeit einzelner Registrierung nicht stabil ist, das bedeutet auch, dass der SCP-Algorithmus eine relativ schlechte Echtzeitfähigkeit hat.

4.7 Vergleich mit dem ICP-Algorithmus

Um ein Überblick von der Leitung des SCP-Algorithmus im Vergleich zu der klassischen lokalen Methode wie ICP-Algorithmus zu bekommen, haben wir auch einen Vergleich durchgeführt. Wir verwendeten die Testeingabedaten für Abschnitt 4.1-4.4 und die default Parameter in der PCL-Bibliothek für den ICP-Algorithmus, wobei setzten wir die maximale Korrespondenz-Entfernung auf 5 Meter.

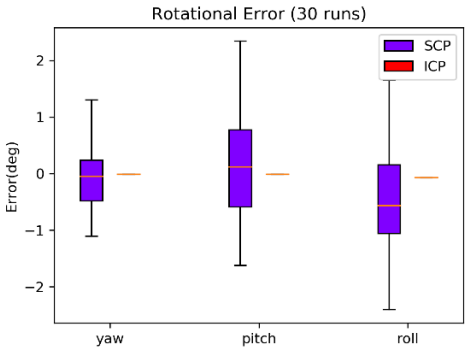

Abb.26. Rotationsfehler von SCP und ICP

Die Testergebnisse werden in Abbildung 26, 27, 28 dargestellt. Wir können sehen, dass im Vergleich zum SCP-Algorithmus der ICP-Algorithmus deutlich kürzere Rechenzeit und deutlich höhere Genauigkeit hat. Die Rotations- und Translationsfehler von ICP-Algorithmus sind jedoch in der Simulation konstant ist. Weil es sich in dem ICP-Algorithmus um keinen zufälligen Prozess handelt.

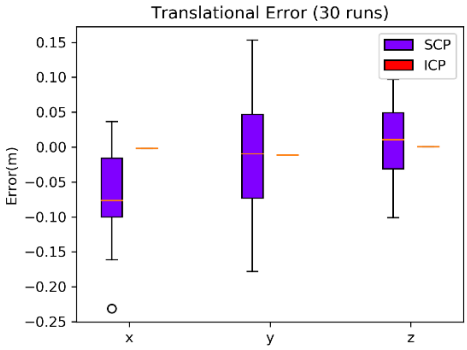

Abb.27. Translationsfehler von SCP und ICP

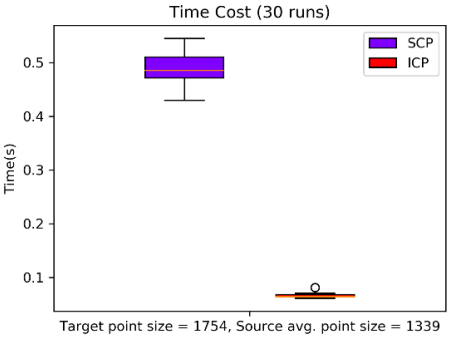

Abb.28. Rechenzeit von SCP und ICP

Bei weiterer Untersuchung nutzten wir die Punktwolken aus dem Punktwolkengenerator, und verglichen die Ergebnisse (siehe Abbildung 29) aus SCP-Algorithmus und ICP-Algorithmus, haben wir jedoch gefunden, dass der ICP-Algorithmus bei größerer Entfernung zwischen den beiden Punktwolken nicht gut konvergieren kann, der SCP-Algorithmus hingegen, kann unabhängig von der Entfernung und Drehung der beiden Punktwolken gut konvergieren.

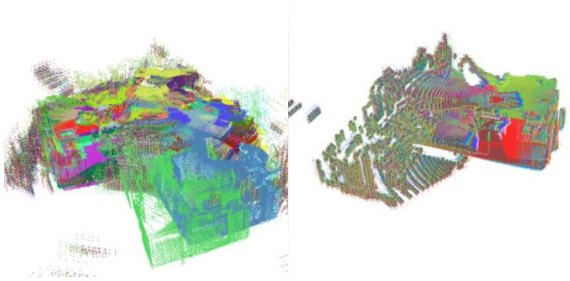

Abb.29. Monte Carlo Simulation von ICP (links) und SCP (rechts) (100 Versuchen)

5. ZUSAMMENFASSUNG

In dieser Arbeit haben wir die Leistung des SCP-Algorithmus ausführlich untersucht.

Wir haben zunächst das Prinzip des SCP-Algorithmus in der originalen Publikation und in der PCL-Bibliothek analysiert, und die mathematischen Hilfsmittel beschrieben.

Dann entwickelten wir eine Software-Testbench für allgemeine Registrierungsalgorithmen für Punktwolken, die mit beliebiger Anzahl von Punktwolkenpaaren, und auch mit beliebiger Anzahl von Registrierungsalgorithmen arbeiten können. Wir haben auch einen Punktwolkengenerator für die Monte Carlo Simulation bei dem Versuchen entwickelt, der eine beliebige Anzahl von Quellpunktwolken mit zufälligen Transformationen, künstlichem Zufallsrauschen und zufälliger Maskierungen erzeugen kann.

In dem Versuch haben wir die Güte der Transformations-schätzung, die Rotations- und Translationsfehler der geschätzten Transformation, die Rechenzeit und die Robustheit des SPC-Algorithmus untersucht und analysiert. Aus den Testergebnissen behaupten wir, dass der SCP-Algorithmus gute Robustheit gegen Parameteränderungen und zufälligem Rauschen hat. Im Vergleich zum Algorithmus mit lokale Optimierungsmethode wie der ICP-Algorithmus weist der SCP-Algorithmus einer relativ geringen Genauigkeit, sowie eine relativ lange und schwankende Rechenzeit beim Versuchen auf. Dennoch ist der SCP-Algorithmus ein globaler Optimierungs-algorithmus, der unabhängig von der Entfernung und Drehung zwischen den beiden Punktwolken gut konvergieren kann. Daher eignet sich der SCP-Algorithmus für die Registrierung von Punktwolken, die nur eine begrenzte Überlagerung der Szene hat, und keine Anforderung an der Echtzeitfähigkeit gibt, sowie das Nähen-Problem von Punktwolken.

Eine andre Möglichkeit, die Reichenzeit und Fehler der Registrierung zu reduzieren, ist, dass wir den SCP-Algorithmus mit lokaler Methode z.B. ICP-Algorithmus kombinieren. Wir könnten den SCP-Algorithmus mit geringen Iterationen nutzen, um eine grobe Transformation zu schätzen, dann folgt mit dem ICP-Algorithmus, die Schätzung zu verfeinern. Damit könnten wir gute Robustheit, hohe Genauigkeit sowie gute Echtzeit-fähigkeit gelichzeitig bekommen.

QUELLEN

[1] PCL-Library, API: pcl::SampleConsensusPrerejective, abgerufen am 06.02.2019, online http://docs.pointclouds.org/trunk/classpcl_1_1_sample_consensus_prerejective.html

[2] A. G. Buch, D. Kraft, J.-K. Kämäräinen, H. G. Petersen and N. Krüger, „Pose Estimation using Local Structure-Specific Shape and Appearance Context", *International Conference on Robotics and Automation (ICRA)*, 2013, online https://ieeexplore.ieee.org/document/6630856

[3] P. J. Besl and N. D. Mckay, „A method for registration of 3-d shapes", *IEEE Transactions Pattern Analysis and Machine Intelligence (PAMI),* vol 14, pp. 239-256, Feb. 1992, online https://ieeexplore.ieee.org/document/121791

[4] Z. Zhang, „Iterative point matching for registration of free-form curve and surface", International Journal of Computer Vision (IJCV), vol. 13, pp.119-152, Oct.1994, online http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.175.770&rep=rep1&type=pdf

[5] Dr.-Ing.S.Dyblenko: „SKRIPT BLR-G-V02 GRUNDLAGEN DER LAGEKINEMATIK (ATTITUDE KINEMATICS), Sommersemester 2018 für Lehrveranstaltungen Bahn- und Lageregelungssysteme für Raumfahrzeuge Lageregelungssysteme für Raumfahrzeuge (MB/MT)"

[6] PCL-Library, „Robust pose estimation of rigid obejcts", online http://pointclouds.org/documentation/tutorials/alignment_prerejective.php

[7] R. B. Rusu, N. Blowdow, and M. Beetz, „Fast point feature histograms (FPFH) for 3D registration ", in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3212 - 3217, May 2009, online https://ieeexplore.ieee.org/document/5152473

[8] PCL-Library, API: pcl::NormalEstimationOMP, abgerufen am 06.02.2019, online http://docs.pointclouds.org/1.8.1/classpcl_1_1_normal_estimation_o_m_p.html

[9] OpenMP, online https://www.openmp.org/

[10] PCL-Library, API: pcl::FPFHEstimationOMP, abgerufen am 06.02.2019, online http://docs.pointclouds.org/1.8.1/classpcl_1_1_f_p_f_h_estimation_o_m_p.html

[11] PCL-Library, Data: room_scan1.pcd/room_scan2.pcd, abgerufen am 06.02.2019, online https://github.com/PointCloudLibrary/data/tree/master/tutorials

[12] PCL-Library, API: pcl::IterativeClosestPoint, abgerufen am 06.02.2019, online http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html