TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik Institut für Automatisierungstechnik

Dokumentation

BESCHREIBUNG DER TESTSOFTWARE

Wintersemester 2018/2019

für Lehrveranstaltung

Oberseminar Automatisierungstechnik

zum Thema

Registrierung von 3D-Punktwolken unter Verwendung der PCL-Bibliothek mit SCP Algorithmus

Zhu, Jinyao

Cao, Bozhi

INHALT

TESTSOFTWARE

1.1 ARCHITEKTUR DER TESTSOFTWARE

1.1.1 Testbench

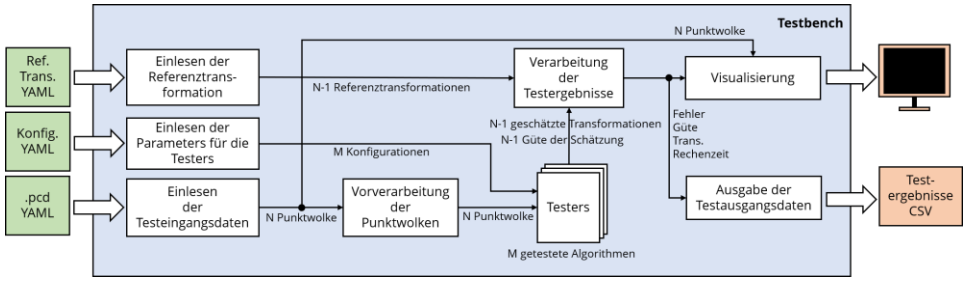

Abbildung 1.1 Architektur der Testsoftware

Abbildung 1.1 zeigt den Aufbau des Testbenchs. Das Testbench startet mit den vom Benutzer angegebenen Konfigurationsdateien. Wir verwenden Konfigurationsdateien im YAML-Format als Eingabe für das Testbench.

Die erste Eingabe (oben) 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 Kandidatenalgorithmen). 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 Translationsfehler, 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.

1.1.2 Punktwolkengenerator

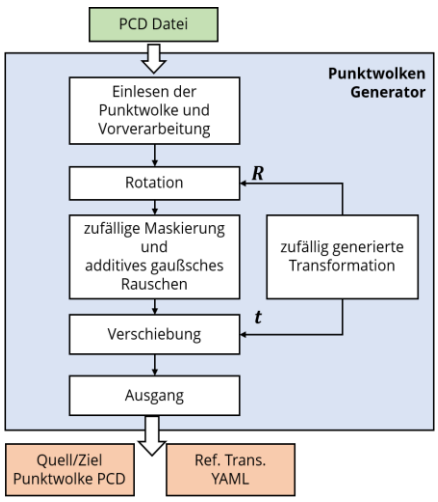

Abbildung 1.2 Architektur der Punktwolkengenerator

Abbildung 1.2 zeigt den Aufbau 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 Schwerpunkt 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.

Nach der Drehung wird eine Ganzzahl aus eine Gleichverteilung 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. 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 (binäre) abgespeichert. Die exakte Transformation wird in einer YAML-Datei geschrieben, die von dem Testbench gelesen werden kann.

1.2 STRUKTUR DER VERZEICHNISSE DER TESTSOFTWARE

OSAT_WS18_SCP/

pcl_test_bench.sln: Visual Studio Solution für die Testsoftware (für weitere Entwicklung).

pcl_test_bench/: Das gesamte Projekt.

- ***doc/***: Dokumentation.
- ***src/***: Quellcode des Testsoftware.
  - *main.cpp*: Main-Funktion, Eingang des Programms.
  - ***lib/***: *yaml-cpp.lib, yaml-cpp_debug.lib.*
  - ***inc/***: Header Dateien für die Testsoftware.
    - ***yaml-cpp/***
    - *test_bench.hpp*: Quellcode des Testbench.
    - *ref_data_generator.hpp*: Quellcode des Punktwolkengenerator.
    - *scp_test.hpp*: Implementierung des SCP-Algorithmus.
    - *icp_test.hpp*: Implementierung des ICP-Algorithmus.
    - *xxx_test.hpp*: Implementierung des anderen Algorithmus.
- ***bin/***:
  - ***ted/***: Testeingangsdaten.
  - ***tad/***: Testausgangsdaten, alle Testergebnisse.
  - ***pcd_scample/***: Einige PCD-Dateien.
  - ***script/***:
    - *run_benchmark.py*: Programm zur Visualisierung der Testergebnisse.
    - *Benchmark.py*: Bestandteil des Python-Modul.
    - *Tester.py*: Bestandteil des Python- Modul.
  - ***cfg/***:
    - *testbench_cfg.yaml*: Konfiguration für das Testbench.
    - *scp_cfg.yaml*: Konfiguration für SCP-Algorithmus.
    - *icp_cfg.yaml*: Konfiguration für ICP-Algorithmus.
    - *xxx_cfg.yaml*: Konfiguration für xxx Algorithmus.
  - *pcl_test_bench.exe*: ausführbare Datei für die Testsoftware.

1.3 TESTEINGANGSDATEN

1.3.1 Punktwolken

Als Testdaten werden Punktwolken *room_scan2.pcd* (Zielpunktwolke) und *room_scan1.pcd* (Quellpunktwolke) [1] zum Beispiel verwendet (entspricht den Eingangsdaten in dem Ordner

pcl_test_bench/bin/ted/ref_room_scan/, siehe *readme.txt* für weitere Details). Die Pfade zu den PCD-Dateien sind in der YMAL-Datei *testbench_cfg.yaml* unter dem Parameter *pcd_files_path* definiert (siehe Abbildung 1.3). Der Typ der Punktwolken ist in diesem Beispiel *pcl::PointXYZ*. Um andere Typen zu nutzen, z.B *pcl::PointXYZRGB*, *pcl::PointXYZI*, müssen die Quellcode dementsprechend modifiziert werden, und dies wird in folgendem Absatz (1.7.4) vorgestellt.

```
pcd_files_path:
    - ted/ref_room_scan/trans_0.pcd  #target point cloud = room_scan2.pcd
    - ted/ref_room_scan/trans_1.pcd
    - ted/ref_room_scan/trans_2.pcd
    - ted/ref_room_scan/trans_3.pcd
    - ted/ref_room_scan/trans_4.pcd
    - ted/ref_room_scan/trans_5.pcd
```

Abbildung 1.3 Definieren Eingangspunktwolken

1.3.2 Referenztransformationen

Die Referenztransformationen werden in der Datei *ref_trans.yaml* aufgezeichnet (die Pfad zur dieser Datei, siehe auch in Absatz 1.6.1 das Parameter *ref_trans_path*). Der Aufbau ist in Abbildung 1.4 gezeigt.

```
1  transformation_1:
2    - 0.7568551
3    - 0.6533574
4    - -0.0174161
5    - -1.6213394
6    - -0.6534376
7    - 0.7569857
8    - 0.0013475
9    - 1.3102897
10   - 0.0140643
11   - 0.0103605
12   - 0.9998482
13   - -0.0517984
14   - 0
15   - 0
16   - 0
17   - 1
18
19 transformation_2:
20   - 0.7568551
21   - 0.6533574
```

Abbildung 1.4 Definieren Referenztransformationen

In dieser Datei werden N (Anzahl der Eingangspunktwolken) Referenztransformationen aufgezeichnet: von *transformation_1* bis *transformation_N*, jede Transformation einer Quellpunktwolke entspricht. Die Elemente der Transformationsmatrix werden nach der Row-Major-Oder von der Transformationsmatrix (4x4) angeordnet, d.h. in der Reihenfolge $t_{11}, t_{12}, t_{13}, t_{14}, t_{21}, \cdots, t_{43}, t_{44}$. Die Definition bzw. die Referenzkoordinatensystem der Transformationsmatrix wird in Abschnitt 1.3.3 vorstellen.

1.3.3 Transformation und Koordinatensystem

Die Definition von der Rotationsmatrix, des Verschiebungsvektors sowie der Transformationsmatrix ist in Absatz 2.2 – 2.4 der Seminararbeit schon erklärt.

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

1.4 AUSGANGSDATEN

1.4.1 Ausgang des Algorithmus

Bei jeder Registrierung erzeugt der Tester (der Registrierungsalgorithmus) eine geschätzte Transformationsmatrix sowie eine Güte der Schätzung. Die Transformationsmatrix hat gleiche Definition wie die in der Seminararbeit (Gleichung 3) definiert.

1.4.2 Ausgang der Testsoftware

Alle Testergebnisse werden in dem Bildschirm dargestellt. Die konkreten Inhalte der Ergebnisse sind in Abbildung 1.5 dargestellt. Die Ergebnisse beinhalten (von oben nach unten): den Namen des Algorithmus, die geschätzte(n) Transformation(en), den Zustand der Konvergenz der Registrierung, die Rotations-/Translationsfehler, die Rechenzeiten, die Punktanzahl sowie den Pfad zu der originalen PCD-Datei der Quellpunktwolke.

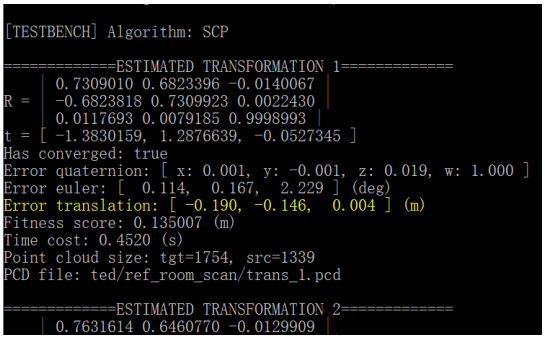

Abbildung 1.5 Testergebnisse auf dem Bildschirm

Am Ende jedes Versuchs werden alle Testergebnisse in eine CSV-Dateien und alle registrierten Punktwolken in PCD-Dateien (binär) abgespeichert. Der Ort zu dem Speichen der Testergebnisse ist in der Konfigurationsdatei ***`testbench_cfg.yaml`*** hinter des Parameters *`result_output_path_root:`* definiert (z.B. *`tad/`*) (vgl. Absatz 1.6.1).

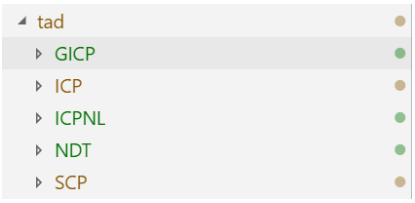

Abbildung 1.6 Unterverzeichnis für Testergebnisse

Für unterschiedlichen Algorithmus speichert die Testsoftware in der Ausgangsordner (default: *`tad/`*) den Testergebnisse in separaten Ordnern unter dem Namen des Algorithmus (vgl. Abbildung 1.6). Der Inhalt der CSV-Datei sieht wie Abbildung 1.7 aus. Die erste Zeile ist der Name des Algorithmus, die zweite die Bedeutung jeder Spalte.

```
1  SCP
2  name,error_yaw,error_pitch,error_roll,error_x,error_y,error_z,converge,time_cost,src_points,tgt_points,fitness,file_p
3  transformation_1,-0.078843,1.560696,-0.278966,-0.054892,0.058486,-0.166567,true,1.431000,1318,1294,0.039801,ted/ref/t
4  transformation_2,0.495230,0.858365,-0.329020,-0.052150,-0.064536,-0.041912,true,0.880000,1133,1294,0.035357,ted/ref/t
5  transformation_3,0.040657,-1.634144,-1.964192,0.005208,0.055407,-0.069609,true,0.468000,1141,1294,0.057235,ted/ref/tr
```

Abbildung 1.7 Die CSV-Datei

Bemerkung: Die in der CSV-Datei abgespeicherte geschätzte Transformation, ist definiert mit einem Verschiebungsvektor und einem Unit-Quaternion (tx, ty, tz, qx, qy, qz, qw). Die Definition des Verschiebungsvektor ist identisch mit der in Seminararbeit (Gleichung 2). Der Unit-Quaternion wird mithilfe der Klasse *`Eigen::Quaternion`* [2] (mit dem Konstruktor *`Quaternion (const MatrixBase< Derived > &other)`*) von der Eigen3-Bibliothek aus der Rotationsmatrix der geschätzten Transformation berechnet.

1.5 VISUALISIERTE DATEN

1.5.1 Visualisierung der Punktwolken

Zur Visualisierung werden alle Punkte in der Quellpunktwolke mit der geschätzten Transformationsmatrix nach dem Referenzkoordinatensystem (S1, Koordinatensystem von der Zielpunktwolke) transformiert (vgl. Gleichung 1):

$$\boldsymbol{p}_i^{S1} = {}_{S2}^{S1}\boldsymbol{T}\boldsymbol{p}_i^{S2}, \qquad i \in 0,1,\cdots,(N-1) \tag{1}$$

wobei N ist die Punkteanzahl der Quellpunktwolke.

Nach der Transformation sind alle Punkte der Quellpunktwolke in Referenzkoordinatensystem (S1) dargestellt, d.h. die Quellpunktwolke wird mit der Zielpunktwolke ausgerichtet (siehe Abbildung 1.8).

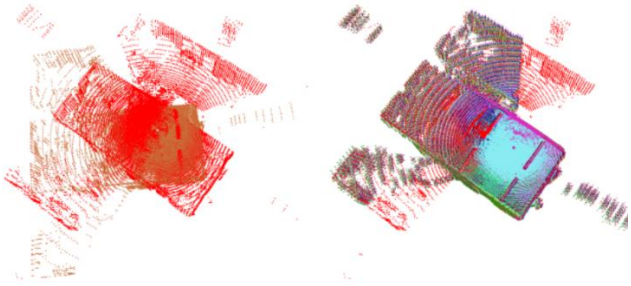

Abbildung 1.8 Ausgerichtete Punktwolken

1.5.2 *Visualisierung der Testergebnisse*

Um die Testergebnisse anschaulicher zu visualisieren, haben wir auch ein Python- Programm entwickelt, das Diagramme aus der Testergebnisse generieren kann. Das Programm befindet sich in dem Ordner *script/*, die Dateien in diesem Ordner ist schon in Absatz 1.2 vorgestellt. Zur Visualisierung müssen Sie den Pfad zu den Testergebnisse sowie den Namen des Algorithmus (in *run_benchmark.py*) geben. Ein Beispiel ist in Abbildung 1.9 gezeigt.

```
1  from Tester import Tester
2  from Benchmark import Benchmark
3
4  tester_names = ['SCP']
5  tester_result_paths = ['../tad/SCP/resultError.csv']
6
```

Abbildung 1.9 Python-Skript(run_benchmark.py), Pfad zur Testergebnisse

Dann führen Sie das Python-Skript durch. Alle Diagramme von den Testergebnisse werden in einen Ordner unter dem gegeben Namen des Algorithmus abgespeichert.

Außerdem kann der Python-Programm mit mehrere Testergebnisse aus unterschiedlichen Algorithmen funktionieren. In diesem Fall wird Benchmark (von Rotations-/Translationsfehler und Rechenzeit) von denen Algorithmen dargestellt. Z.B. führen wir Registrierung mit mehreren Algorithmen (SCP, ICP, ICP NL, GICP, NDT) für gleiches Dataset durch. Die Testergebnisse sind in der Ordner *tad/* in separatem Ordner gespeichert. Wir ändern die Pfade zu den Testergebnisse im Python-Skript dementsprechend (siehe Abbildung 1.10).

```
# given names of the testers
tester_names = ['SCP','ICP','ICPNL','GICP','NDT']
# csv files path of each tester
tester_result_paths = ['../tad/SCP/resultError.csv',
                       '../tad/ICP/resultError.csv',
                       '../tad/ICPNL/resultError.csv',
                       '../tad/GICP/resultError.csv',
                       '../tad/NDT/resultError.csv']
```

Abbildung 1.10 Python-Skript, Visualisierung von mehreren Algorithmen

Dann können wir Benchmark von den Algorithmen bekommen. Abbildung 1.11 zeigt zum Beispiel die Rotationsfehler.

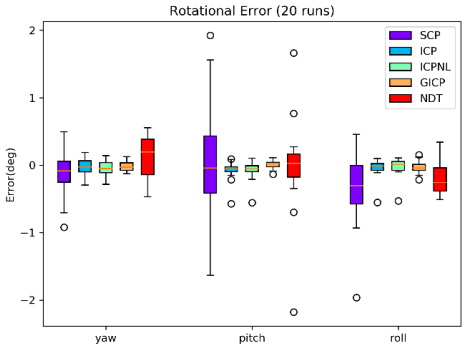

Abbildung 1.11 Benchmark

1.6 PARAMETER

1.6.1 Parameter für das Testbench

Alle Parameter für das Testbench sind in der Datei *testbench_cfg.yaml* (in *cfg/* siehe auch Absatz 1.2) definiert. Es gibt folgenden Parameters in dieser Datei:

- *grid_filter_leafsize*: leaf size für das Down-Sampling beim Vorverarbeitungsschritt des Testbenchs. (Meter, > 0)
- *scp_cfg_path*: Pfad zur Konfiguration für SCP-Algorithmus (*scp_cfg.yaml*).
- *has_ref_transformation* (true/false): ob wir Referenztransformation haben(benutzen), wenn NEIN, wird das Parameter *ref_trans_path* unbenutzt, und gibt das Testbench auch nur visuelle Ergebnis, d.h. wird keine Schätzungsfehler berechnet.
- *ref_trans_path*: Pfade zu den Referenztransformationen (*ref_trans.yaml*) (vgl. Abschnitt 1.3.2).
- *result_output_path_root*: Speicherort aller Testergebnisse.
- *pcd_files_path*: Pfade zu den Eingangspunktwolken (vgl. Abschnitt 1.3.1).

1.6.2 Parameter für den SCP-Algorithmus

Die Dateien *scp_cfg.yaml* beinhaltet alle Konfigurationen für den SCP-Algorithmus. Man kann folgende Parameter einstellen, diese Parameter sind auch in Abschnitt 3.3 der Seminararbeit vorgestellt:

- *norm_est_radius_search*: Search-Radius für den Norm-Estimator. (Meter, > 0, Typ. 3~4 x leaf size des Grid-Filters)
- *fpfh_est_radius_search*: Search-Radius für den FPFH-Estimator. (Meter, >0, Typ. 3~4 x leaf size des Grid-Filters)
- *scp_max_corr_distance*: max. correspondence distance. (Meter, >0, Typ. 2.5 x leaf size des Grid-Filters)
- *scp_max_iter*: max. iterations. (≥ 0)
- *scp_sim_threshold*: similarity threshold. ($\in [0,1)$)
- *scp_corr_randomness*: correspondence randomness. (≥ 1)
- *scp_inlier_fraction*: inlier fraction. ($\in (0,1)$)
- *scp_number_of_samples*: number of samples. (≥ 3)

Hinweis: Sie müssen nicht nach jeder Änderung der Parameter die Software neustarten. Die Software wird vor jedem Testen die Parameter neu aufladen.

1.7 IMPLEMENTIERUNG DES ALGORITHMUS

1.7.1 Testbench

Die konkrete Implementierung des Testbench ist in der Datei *test_bench.hpp* in dem Ordner *src/inc/*. Dort wird die Klasse *Testbench* definiert, welche ein Klassentemplate ist, d.h. sie kann mit beliebigem Punkt-Typ funktionieren, solang der Typ in der PCL-Bibliothek schon definiert ist. Beim Entwurf entkoppelten wir das Testbench gegen die Registrierungsalgorithmen, so dass das Testbench für meisten Registrierungsalgorithmen aus der PCL-Bibliothek funktionieren kann. Und die Klasse *Testbench* ist als eine Basisklasse dient. Die Implementierungen unterschiedlicher Registrierungs-algorithmen wird von dieser Klasse abgeleitet (derived class).

1.7.2 SCP-Algorithmus

Die konkrete Implementierung des SCP-Algorithmus befindet sich in dem Ordner *src/inc/* unter dem Namen von *scp_test.hpp*.

In *scp_test.hpp* wird die Klasse *ScpTest* definiert. Diese Klasse ist abgeleitet (derived class) von der Basisklasse *Testbench*. Die detaillierten Beschreibungen zu jeder Funktion sind wie folgt:

- *ScpTest()* (Zeile 23): Konstruktor der Klasse, Name des Algorithmus setzen, Initialisierung für den Algorithmus.

- *loadCandidateConfig()* (Zeile 32): in dieser Funktion werden alle Parameter für den SCP-Algorithmus aus der YAML-Konfigurationsdatei eingelesen. Dieser Funktion wird von dem Testbench aufgerufen.
- *doAlignOnce()* (Zeile 77): konkrete Implementierung des SCP-Algorithmus für eine Registrierung. Eingangsdaten sind die Quell-/und Zielpunktwolke (*src_cloud* und *tgt_cloud*), und die Ausgangsdaten ist die geschätzte Transformationsmatrix (*transform*) und die Konvergenzzustand (type *bool*, Rückgabewert der Funktion). Dieser Funktion wird von dem Testbench aufgerufen.
- *getFitness()*(Zeile 114): Fitness-Score von der Registrierung rückgeben. Dieser Funktion wird von dem Testbench aufgerufen.

1.7.3 Main-Funktion

Der Eingang für die Testsoftware ist die Datei *main.cpp* in dem Ordner *src/*. Um ein Test für einen Registrierungsalgorithmus auf das Testbench durchzuführen, braucht hier 3 Codezeilen, z.B. für den SCP-Algorithmus Zeile 103 ~ 105 (siehe Abbildung 1.11).

```
102     else if (user_input == "2") {
103         ScpTest<PointType> test_bench = ScpTest<PointType>();
104         test_bench.loadTestBenchConfig("cfg/testbench_cfg.yaml");
105         test_bench.runTestBench();
106     }
```

Abbildung 1.11 Main-Funktion

1.7.4 Testen mit anderen Punktwolkentypen

In der Seminararbeit wird der Typ *pcl::PointXYZ* für alle Punktwolken verwendet. Tatsächlich kann der SCP-Algorithmus (abhängig von dem FPFH-Estimator) mit mehreren Punktwolkentypen funktionieren (zumindestens haben wir mit *pcl::PointXYZRGB*, *pcd::PointXYZI* getestet). In diesem Fall müssen Sie die Typdefinition in der Datei *main.cpp* (in *src/*) modifizieren. In Zeile 14 von *main.cpp* die Typdefinition für *PointType* ändern. Dabei müssen Sie allerdings die Testsoftware neu kompilieren. Zum neuen Kompilieren lesen Sie die Portierungsanleitung in Absatz 1.9.2.

1.7.5 Hinzufügen weitere Registrierungsalgorithmen

Wie in der Seminararbeit erwähnt, ist das Testbench für allgemeine bzw. normale Registrierungs-algorithmen ausgelegt, es ist sich nicht auf den SCP-Algorithmus beschränkt. Wenn Sie weitere Algorithmen in das Testbench hinzufügen wollen, lesen Sie die Datei *xxx_test.hpp* (in *src/inc/*). Sie ist eine Vorlage für den Tester (also den Kandidatenalgorithmus), und hat gleiche Grundstruktur wie die *scp_test.hpp*. Erfüllen Sie die Funktionen darin nach den Hinweisen, benennen Sie ihr nach dem Namen Ihres Algorithmus, und in demselben Ordner (*src/inc/*) legen. Dann modifizieren Sie das „Menü“ in *main.cpp*, und erstellen Sie neu Objekt entsprechend dem neuen Algorithmus darin (folgen Sie den Hinweisen in der Quellcode). Zuletzt kompilieren Sie die Software neu. Zum neuen Kompilieren lesen Sie die Portierungsanleitung in Absatz 1.9.2.

1.8 TESTABLAUF

1.8.1 Testbench

Schritt 1: Modifizieren Sie die YAML-Dateien (in *cfg/* Ordner), die in Absatz 1.6 vorgestellt ist. Geben Sie die Pfade zu den PCD-Dateien, zu der Referenztransformationen (wenn Sie die exakten Schätzungsfehler brauchen). Setzen Sie die Parameter für das Testbench und den SCP-Algorithmus jeweils in *test_bench.yaml* und *scp_test.yaml* (siehe Abbildung 1.11). Zum Testzweck können Sie zunächst die vorgegebenen Parameter verwenden, dann brauchen Sie bei diesem Schritt kein Modifizieren.

Abbildung 1.11 Schritt 1, Konfiguration

Schritt 2: Führen Sie die Testsoftware mit *`pcl_test_bench.exe`* (in *`bin/`*) durch. Die Ansicht der Testsoftware ist in Abbildung 1.12 gezeigt.

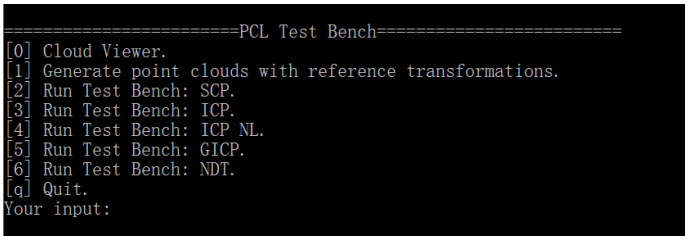

Abbildung 1.12 Schritt 2, Funktion wählen

Dann geben Sie „2“ ein (für den SCP-Algorithmus. Sie können auch andren Algorithmus testen, wir konzentrieren uns aber hier nur auf den SCP-Algorithmus), und drucken „Enter“ auf Ihrer Tastatur. Die Software wird die benötigen Konfigurationsdateien sowie PCD-Dateien automatisch aufladen. Die Konsole sollte jetzt wie in Abbildung 1.13 aussehen, die PCLVisualizer wird gleichzeitig auf dem Bildschirm angezeigt (siehe Abbildung 1.14). Auf dem PCLVisualizer können Sie das Rad Ihrer Maus (mittlere Taste) verwenden, um die Punktwolken zu vergrößern bzw. verkleinern.

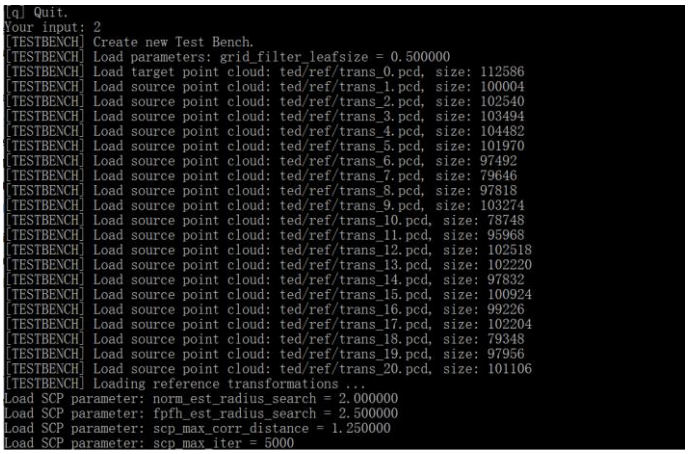

Abbildung 1.13 Schritt 2, Konfigurationen aufladen

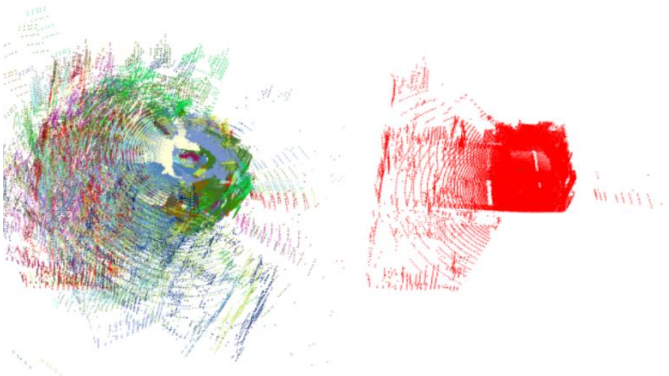

Abbildung 1.14 Schritt 2, PCLVisualizer

Schritt 3: Aktivieren Sie das Fenster des PCLVisualizer, also einfach per Klicken auf dem Fenster. Und drucken Sie „q“, um die Registrierung zu starten. Die Testsoftware sollte jetzt alle Quellpunktwolken mit

der Zielpunktwolke registrieren. Die aktuell gearbeiteten Punktwolken werden auf PCLVisualizer dargestellt.

Schritt 4: Nach den Registrierungen werden alle Testergebnisse auf der Konsole dargestellt (vgl. Abbildung 1.15). Die Testergebnisse werden also automatisch in ein CSV-Datei in dem von Ihnen spezifischen Ordner (default: *tad/*) abgespeichert, wenn Sie exakte Referenztransformationen nutzen (siehe Absatz 1.6.1).

```
[TESTBENCH] Algorithm: SCP

============ESTIMATED TRANSFORMATION 1============
     | 0.5762170 0.8168564 -0.0268240 |
R =  | -0.8017633 0.5713296 0.1753799 |
     | 0.1585856 -0.0795504 0.9841354 |
t = [ 0.2467514, -0.3608380, -0.1456770 ]
Has converged: true
Error quaternion: [ x: 0.014, y: 0.001, z: -0.002, w: 1.000 ]
Error euler: [ -0.079,  1.561, -0.279 ] (deg)
Error translation: [ -0.055,  0.058, -0.167 ] (m)
Fitness score: 0.039801 (m)
Time cost: 1.4310 (s)
Point cloud size: tgt=1294, src=1318
PCD file: ted/ref/trans_1.pcd

============ESTIMATED TRANSFORMATION 2============
     | 0.9182385 -0.3843065 0.0956426 |
```

Abbildung 1.15 Schritt 4, Testergebnisse auf dem Bildschirm

Schritt 5: Wenn Sie detaillierte Diagramm mit Python-Programm generieren wollen, setzen Sie die Pfade zu den Testergebnisse, wie das in Abschnitt 1.5.2 vorgestellt ist. Dann führen Sie das Python-Skript *run_test_bench.py* durch, alle Diagramme von den Testergebnisse werden in separaten Ordnern generiert (vgl. Abbildung 1.16).

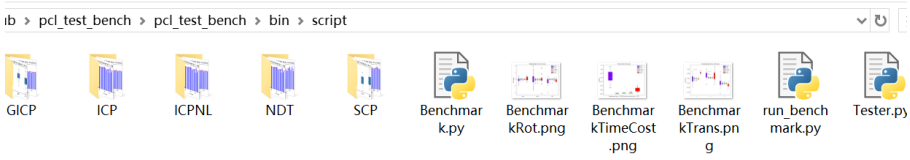

Abbildung 1.16 Python-Skript, Diagramme generieren

1.8.2 Punktwolkengenerator (optional)

Wenn Sie der in die Seminararbeit (Absatz 3.2) vorgestellte Punktwolkengenerator benutzen wollen, befolgen Sie folgende Schritte:

Schritt 1: Wie beim Testablauf von Testbench, führen Sie die ausführbare Datei *pcl_test_bench.exe* im *bin/* Ordner durch, und geben Sie „1“ ein (vgl. Abbildung 1.17).

```
====================PCL Test Bench====================
[0] Cloud Viewer.
[1] Generate point clouds with reference transformations.
[2] Run Test Bench: SCP.
[3] Run Test Bench: ICP.
[4] Run Test Bench: ICP NL.
[5] Run Test Bench: GICP.
[6] Run Test Bench: NDT.
[q] Quit.
Your input: 1
Target .pcd/.ply file(eg: bun0.pcd bun0.ply): pcd_sample/room_scan2.pcd
Output path(eg: ted/ref/): ted/ref/
Number of transformations: 30
Maximal translation x(m): 0.5
Maximal translation y(m): 0.5
Maximal translation z(m): 0.5
Maximal rotation angle yaw(deg): 90
Maximal rotation angle pitch(deg): 10
Maximal rotation angle roll(deg): 10
Add noise to point clouds?(y/n):y
noise std(m):0.05
Add noise ...
Generate transformation1=
  0.995754   0.0636354   0.0665213   -0.366568
-0.0700466    0.992622   0.0989644   -0.142753
-0.0597329   -0.103204    0.992865    0.134276
         0          -0          -0           1
```

Abbildung 1.17 Punktwolkengenerator

Schritt 2: Geben Sie benötige Parameter nach den Hinweisen auf der Konsole ein. Z.B. Pfad der originalen Punktwolke (als Zielpunktwolke), Pfad des Ordners zur Speicherung der generierten PCD-Dateien sowie der Referenztransformationen, Anzahl der zu erzeugenden Punktwolken, etc. Nach der Generierung könnte der Ausgabeordner (hier *ted/ref/*) wie Abbildung 1.18 aussehen.

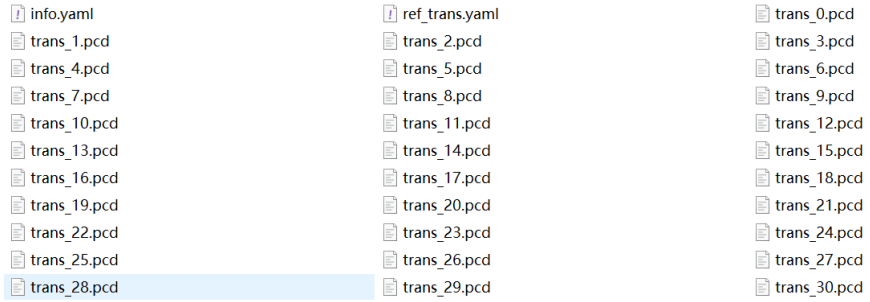

Abbildung 1.18 Ausgabeordner des Punktwolkengenerator

Wobei *info.yaml* beinhaltet alle Parameter über die generierten Punktwolken (z.B. die Pfad zur originalen Punktwolke, Parameter für die Rauschen, sowie die Namen der generierten PCD-Dateien). Die Datei *ref_trans.yaml* beinhaltet die exakten Transformationsmatrizen, die in Absatz 1.3.2 vorgestellt sind.

1.9 INSTALLATION, PORTIERUNG UND PC-ANFORDERUNGEN

Tabelle 1.1 Erforderliche PC-Konfigurationen

für Ausführen (via Installation)	**für Weiterentwicklung (via Portierung)**
Allgemeine Konfiguration	
CPU: 1.8-GHz-Prozessor oder schneller.	CPU: 1.8-GHz-Prozessor oder schneller.
RAM: 2 GB RAM; 4 GB RAM empfohlen.	RAM: 2 GB RAM; 4 GB RAM empfohlen.
Windows 7, 8.1, 10, 64bit	Windows 7, 8.1, 10, 64bit
Spezielle aufgabenrelevante Konfiguration	
Python 3.7 mit *numpy* und *matplotlib* (optional)	Microsoft Visual Studio Community 2017 (C++)

1.9.1 Installationsanleitung für die Testsoftware

Zur Installation der Testsoftware einfach kopieren Sie die Archiv-Datei (Zip-Datei) auf dem Datenträger in Ihren Computer, und dekomprimieren die Zip-Datei. Dann können Sie die Testsoftware mit der ausführbaren Datei *pcl_test_bench.exe* (im *bin/* Ordner) starten.

1.9.2 Installation von Python3.7 (optional)

Laden Sie Python 3.7 mit dieser Seite: https://www.python.org/downloads/ herunter, und installieren. Bei der Installation machen Sie sicher, dass die „pip“ gewählt wird (vgl. Abbildung 1.19).

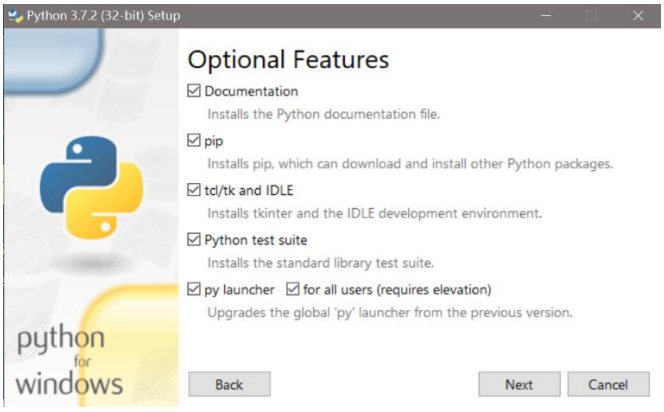

Abbildung 1.19 Python installieren

Um *numpy* und *matplotlib* zu installieren, öffnen Sie die Windows PowerShell (oder drücken „Win“ + „R“ dann geben *cmd* ein und klicken „bestätigen“), dann geben folgende Befehle ein:

```
pip install numpy
pip install matplotlib
```

1.9.3 Portierungsanleitung für die Testsoftware

Zur weiteren Entwicklung müssen Sie die PCL-1.8.1 (vielleicht höhere Version kann auch funktionieren) in Ihren Computer installieren. Sie können die vorkompilierte Version von der PCL-Bibliothek auf dieser Seite herunterladen: https://github.com/PointCloudLibrary/pcl/releases. Zu den detaillierten Einstellungen für die Installation der PCL-Bibliothek lesen Sie Absatz 1-2 der Anleitung (*setup.html*) im Ordner *doc/* durch.

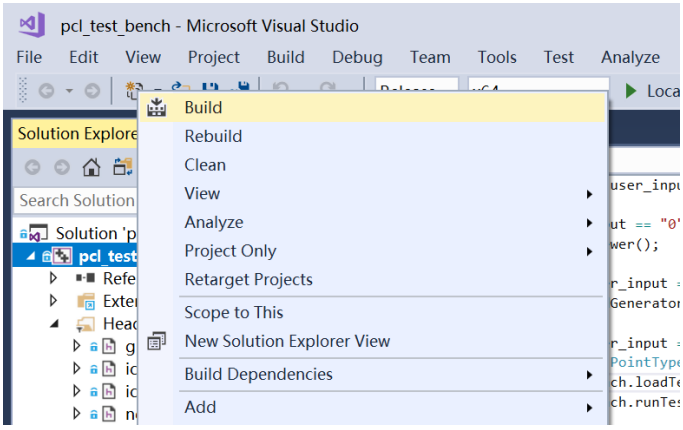

Abbildung 1.20 Build Projekt

Nach der Installation öffnen Sie die vorhandene „Solution“ (*pcl_test_bench.sln*), klicken mit der rechten Maustaste auf dem Projekt (vgl. Abbildung 1.20). Falls die „Solution“ nicht funktionieren kann, können Sie die *setup.html* weiterlesen, und erstellen Sie selbst Ihre Entwicklungsumgebung für die PCL-Bibliothek.

1.10 QUELLEN

[1] *room_scan1.pcd* / *room_scan2.pcd*, abgerufen am 06.02.2019, online https://github.com/PointCloudLibrary/data/tree/master/tutorials

[2] Eigen 3.3.7, *class Eigen::Quaternion*, abgerufen am 06.02.2019, online https://eigen.tuxfamily.org/dox/classEigen_1_1Quaternion.html#aba694e0bbafcf5554b77cfe8ce69c14a