RobCodeGenerator

Erzeugt von Doxygen 1.9.7

1 Beschreibung Roboter Path Editor 1
1.1 Nutzen . . . 1
1.2 Aufbau . . . 1
1.2.1 Logging . . . 1
1.2.2 Daten einlesen . . . 1
1.2.3 Daten verarbeiten . . . 1
1.2.4 Roboter Code erstellen . . . 1

2 Hierarchie-Verzeichnis 3
2.1 Klassenhierarchie . . . 3

3 Klassen-Verzeichnis 5
3.1 Auflistung der Klassen . . . 5

4 Datei-Verzeichnis 7
4.1 Auflistung der Dateien . . . 7

5 Klassen-Dokumentation 9
5.1 CEulerMatrix Klassenreferenz . . . 9
5.1.1 Ausführliche Beschreibung . . . 9
5.1.2 Beschreibung der Konstruktoren und Destruktoren . . . 10
5.1.2.1 CEulerMatrix() [1/2] . . . 10
5.1.2.2 CEulerMatrix() [2/2] . . . 10
5.1.2.3 ~CEulerMatrix() . . . 10
5.1.3 Dokumentation der Elementfunktionen . . . 11
5.1.3.1 angels2mat() . . . 11
5.1.3.2 calculateAngels() . . . 11
5.1.3.3 getEulerMatrix() . . . 12
5.1.3.4 getMatrix() . . . 12
5.1.3.5 setMatrix() . . . 13
5.1.4 Dokumentation der Datenelemente . . . 13
5.1.4.1 eulerMatrix . . . 13
5.2 CGUI Klassenreferenz . . . 13
5.2.1 Ausführliche Beschreibung . . . 13
5.2.2 Beschreibung der Konstruktoren und Destruktoren . . . 14
5.2.2.1 CGUI() . . . 14
5.2.2.2 ~CGUI() . . . 14
5.3 CInputParameter Klassenreferenz . . . 14
5.3.1 Ausführliche Beschreibung . . . 15
5.3.2 Beschreibung der Konstruktoren und Destruktoren . . . 15
5.3.2.1 CInputParameter() [1/2] . . . 15
5.3.2.2 CInputParameter() [2/2] . . . 15
5.3.2.3 ~CInputParameter() . . . 16
5.3.3 Dokumentation der Elementfunktionen . . . 16

5.3.3.1 detectJump() . . . 16
5.3.3.2 getAngles() . . . 17
5.3.3.3 getOrientationManual() . . . 17
5.3.3.4 getPath() . . . 18
5.3.3.5 getSpeed() . . . 18
5.3.3.6 getSpeedManual() . . . 18
5.3.3.7 openFile() . . . 18
5.3.3.8 setOrientation() . . . 19
5.3.3.9 setSpeed() . . . 20
5.3.4 Dokumentation der Datenelemente . . . 20
5.3.4.1 A . . . 20
5.3.4.2 B . . . 20
5.3.4.3 C . . . 20
5.3.4.4 difference . . . 21
5.3.4.5 initialPath . . . 21
5.3.4.6 orientationManual . . . 21
5.3.4.7 speed . . . 21
5.3.4.8 speedManual . . . 21
5.4 CInputPoint3D Klassenreferenz . . . 22
5.4.1 Ausführliche Beschreibung . . . 23
5.4.2 Beschreibung der Konstruktoren und Destruktoren . . . 23
5.4.2.1 CInputPoint3D() [1/2] . . . 23
5.4.2.2 CInputPoint3D() [2/2] . . . 24
5.4.2.3 ∼CInputPoint3D() . . . 24
5.4.3 Dokumentation der Elementfunktionen . . . 24
5.4.3.1 getEulerMatrix() . . . 24
5.4.3.2 getTime() . . . 25
5.4.3.3 setEulerMatrix() . . . 25
5.4.3.4 setPoint() . . . 25
5.4.3.5 setTime() . . . 26
5.4.4 Dokumentation der Datenelemente . . . 26
5.4.4.1 orientationMatrix . . . 26
5.4.4.2 timestamp . . . 26
5.5 CLine3D Klassenreferenz . . . 26
5.5.1 Ausführliche Beschreibung . . . 27
5.5.2 Beschreibung der Konstruktoren und Destruktoren . . . 27
5.5.2.1 CLine3D() [1/2] . . . 27
5.5.2.2 CLine3D() [2/2] . . . 27
5.5.2.3 ∼CLine3D() . . . 28
5.5.3 Dokumentation der Datenelemente . . . 28
5.5.3.1 p1 . . . 28
5.5.3.2 p2 . . . 28

5.6 CLogging Klassenreferenz . . . 28
5.6.1 Ausführliche Beschreibung . . . 29
5.6.2 Beschreibung der Konstruktoren und Destruktoren . . . 29
5.6.2.1 CLogging() [1/2] . . . 29
5.6.2.2 CLogging() [2/2] . . . 29
5.6.2.3 ~CLogging() . . . 30
5.6.3 Dokumentation der Elementfunktionen . . . 30
5.6.3.1 logData() [1/2] . . . 30
5.6.3.2 logData() [2/2] . . . 30
5.6.3.3 setStep() . . . 31
5.6.4 Dokumentation der Datenelemente . . . 32
5.6.4.1 path . . . 32
5.6.4.2 step . . . 32
5.7 CMeanFilter Klassenreferenz . . . 32
5.7.1 Ausführliche Beschreibung . . . 33
5.7.2 Beschreibung der Konstruktoren und Destruktoren . . . 33
5.7.2.1 CMeanFilter() [1/2] . . . 33
5.7.2.2 CMeanFilter() [2/2] . . . 33
5.7.2.3 ~CMeanFilter() . . . 34
5.7.3 Dokumentation der Elementfunktionen . . . 34
5.7.3.1 calculateMean() . . . 34
5.7.3.2 getPath() . . . 35
5.7.3.3 getWindowSize() . . . 35
5.7.3.4 mean() . . . 35
5.7.3.5 setWindowSize() . . . 36
5.7.4 Dokumentation der Datenelemente . . . 36
5.7.4.1 meanPath . . . 36
5.7.4.2 windowSize . . . 36
5.8 COutputPoint3D Klassenreferenz . . . 37
5.8.1 Ausführliche Beschreibung . . . 38
5.8.2 Beschreibung der Konstruktoren und Destruktoren . . . 38
5.8.2.1 COutputPoint3D() [1/2] . . . 38
5.8.2.2 COutputPoint3D() [2/2] . . . 39
5.8.2.3 ~COutputPoint3D() . . . 39
5.8.3 Dokumentation der Elementfunktionen . . . 40
5.8.3.1 getA() . . . 40
5.8.3.2 getB() . . . 40
5.8.3.3 getC() . . . 40
5.8.3.4 getSpeed() . . . 41
5.8.3.5 setA() . . . 41
5.8.3.6 setB() . . . 41
5.8.3.7 setC() . . . 41

5.8.3.8 setSpeed() . . . 43
5.8.4 Dokumentation der Datenelemente . . . 43
5.8.4.1 a . . . 43
5.8.4.2 b . . . 43
5.8.4.3 c . . . 43
5.8.4.4 speed . . . 44
5.9 CPathBuilder Klassenreferenz . . . 44
5.9.1 Ausführliche Beschreibung . . . 44
5.9.2 Beschreibung der Konstruktoren und Destruktoren . . . 45
5.9.2.1 CPathBuilder() . . . 45
5.9.2.2 ~CPathBuilder() . . . 45
5.9.3 Dokumentation der Elementfunktionen . . . 45
5.9.3.1 createPath() . . . 45
5.9.3.2 getPath() . . . 46
5.9.4 Dokumentation der Datenelemente . . . 46
5.9.4.1 path . . . 46
5.10 CPoint3D Klassenreferenz . . . 46
5.10.1 Ausführliche Beschreibung . . . 47
5.10.2 Beschreibung der Konstruktoren und Destruktoren . . . 48
5.10.2.1 CPoint3D() [1/2] . . . 48
5.10.2.2 CPoint3D() [2/2] . . . 48
5.10.2.3 ~CPoint3D() . . . 48
5.10.3 Dokumentation der Elementfunktionen . . . 49
5.10.3.1 distanceTo() [1/2] . . . 49
5.10.3.2 distanceTo() [2/2] . . . 49
5.10.3.3 getX() . . . 50
5.10.3.4 getY() . . . 50
5.10.3.5 getZ() . . . 50
5.10.3.6 set() . . . 51
5.10.3.7 setX() . . . 51
5.10.3.8 setY() . . . 51
5.10.3.9 setZ() . . . 52
5.10.4 Dokumentation der Datenelemente . . . 52
5.10.4.1 x . . . 52
5.10.4.2 y . . . 52
5.10.4.3 z . . . 52
5.11 CRobCodeGenerator Klassenreferenz . . . 52
5.11.1 Ausführliche Beschreibung . . . 53
5.11.2 Beschreibung der Konstruktoren und Destruktoren . . . 53
5.11.2.1 CRobCodeGenerator() [1/2] . . . 53
5.11.2.2 CRobCodeGenerator() [2/2] . . . 54
5.11.2.3 ~CRobCodeGenerator() . . . 54

5.11.3 Dokumentation der Elementfunktionen . . . 54
5.11.3.1 calculateAngles() . . . 54
5.11.3.2 calculateSpeed() . . . 55
5.11.3.3 generateRobCode() . . . 55
5.11.3.4 postProcessing() . . . 56
5.11.4 Dokumentation der Datenelemente . . . 57
5.11.4.1 A . . . 57
5.11.4.2 B . . . 57
5.11.4.3 C . . . 57
5.11.4.4 orientationManual . . . 58
5.11.4.5 processedPath . . . 58
5.11.4.6 speed . . . 58
5.11.4.7 speedManual . . . 58
5.12 CSegmentApproximator Klassenreferenz . . . 58
5.12.1 Ausführliche Beschreibung . . . 59
5.12.2 Beschreibung der Konstruktoren und Destruktoren . . . 59
5.12.2.1 CSegmentApproximator() . . . 59
5.12.2.2 ∼CSegmentApproximator() . . . 60
5.12.3 Dokumentation der Elementfunktionen . . . 60
5.12.3.1 approx() . . . 60
5.12.3.2 douglasPeuckerRecursive() . . . 60
5.12.3.3 getmaxDistance() . . . 61
5.12.3.4 getSegmentsApproxVector() . . . 61
5.12.3.5 setmaxDistance() . . . 62
5.12.4 Dokumentation der Datenelemente . . . 62
5.12.4.1 maxDistance . . . 62
5.12.4.2 segmentsApprox . . . 62

6 Datei-Dokumentation **63**
6.1 header/EulerMatrix.h-Dateireferenz . . . 63
6.1.1 Ausführliche Beschreibung . . . 63
6.2 EulerMatrix.h . . . 63
6.3 header/GUI.h-Dateireferenz . . . 64
6.4 GUI.h . . . 64
6.5 header/InputParameter.h-Dateireferenz . . . 64
6.5.1 Ausführliche Beschreibung . . . 64
6.6 InputParameter.h . . . 65
6.7 header/Line3D.h-Dateireferenz . . . 65
6.7.1 Ausführliche Beschreibung . . . 65
6.8 Line3D.h . . . 66
6.9 header/Logging.h-Dateireferenz . . . 66
6.9.1 Ausführliche Beschreibung . . . 66

6.10 Logging.h . . . 67
6.11 header/MeanFilter.h-Dateireferenz . . . 67
6.11.1 Ausführliche Beschreibung . . . 67
6.12 MeanFilter.h . . . 68
6.13 header/PathBuilder.h-Dateireferenz . . . 68
6.13.1 Ausführliche Beschreibung . . . 68
6.14 PathBuilder.h . . . 69
6.15 header/Point3D.h-Dateireferenz . . . 69
6.15.1 Ausführliche Beschreibung . . . 69
6.16 Point3D.h . . . 70
6.17 header/RobCodeGenerator.h-Dateireferenz . . . 70
6.17.1 Ausführliche Beschreibung . . . 71
6.17.2 Makro-Dokumentation . . . 71
6.17.2.1 MAX_SPEED . . . 71
6.18 RobCodeGenerator.h . . . 71
6.19 header/SegmentApproximator.h-Dateireferenz . . . 72
6.19.1 Ausführliche Beschreibung . . . 72
6.20 SegmentApproximator.h . . . 72
6.21 src/EulerMatrix.cpp-Dateireferenz . . . 73
6.21.1 Ausführliche Beschreibung . . . 73
6.22 EulerMatrix.cpp . . . 73
6.23 src/GUI.cpp-Dateireferenz . . . 74
6.24 GUI.cpp . . . 74
6.25 src/InputParameter.cpp-Dateireferenz . . . 74
6.25.1 Ausführliche Beschreibung . . . 75
6.26 InputParameter.cpp . . . 75
6.27 src/Line3D.cpp-Dateireferenz . . . 76
6.27.1 Ausführliche Beschreibung . . . 76
6.28 Line3D.cpp . . . 77
6.29 src/Logging.cpp-Dateireferenz . . . 77
6.29.1 Ausführliche Beschreibung . . . 77
6.30 Logging.cpp . . . 77
6.31 src/MeanFilter.cpp-Dateireferenz . . . 79
6.31.1 Ausführliche Beschreibung . . . 79
6.32 MeanFilter.cpp . . . 79
6.33 src/PathBuilder.cpp-Dateireferenz . . . 80
6.33.1 Ausführliche Beschreibung . . . 80
6.34 PathBuilder.cpp . . . 80
6.35 src/Point3D.cpp-Dateireferenz . . . 81
6.35.1 Ausführliche Beschreibung . . . 81
6.36 Point3D.cpp . . . 81
6.37 src/RobCodeGenerator.cpp-Dateireferenz . . . 84

6.37.1 Ausführliche Beschreibung 84
6.38 RobCodeGenerator.cpp 84
6.39 src/RobPathEditor.cpp-Dateireferenz 86
6.39.1 Ausführliche Beschreibung 86
6.39.2 Dokumentation der Funktionen 87
6.39.2.1 main() 87
6.40 RobPathEditor.cpp 87
6.41 src/SegmentApproximator.cpp-Dateireferenz 88
6.41.1 Ausführliche Beschreibung 88
6.42 SegmentApproximator.cpp 89

Index **91**

Kapitel 1

Beschreibung Roboter Path Editor

1.1 Nutzen

Mit diesem Programm sollen händisch aufgenommene Pfad Daten einer Roboterbewegung zu einem für Kuka Roboter lesbaren File gemacht werden. Zusätzlich soll einstellbar sein ob die Orientierung berechnet werden soll, oder eingegeben werden soll. Das selbe gilt für Geschwindigkeitsdaten.

1.2 Aufbau

In der Grundidee werden die eingelesenen Daten immer aus der vorhergegangenen Klasse ausgelesen und nach der Verarbeitung in der aktuellen Klasse gespeichert.

1.2.1 Logging

Zuerst wird die Loggingklasse CLogging initialisiert. In ihr wird gespeichert in welchem Schritt das Programm gerade ist. Dieser Klasse wird ein Pfad übergeben an welchem die Daten gespeichert werden sollen.

1.2.2 Daten einlesen

Als nächstes werden die Nutzerdaten eingelesen und anschliessend die aufgenommenen Daten eingelesen. Dabei wird Überprüft ob es sich um einen zusammenhängenden Pfad handelt. Das passiert in der Klasse CInputParameter.

1.2.3 Daten verarbeiten

In mehreren Schritten folgt eine Nachbearbeitung der Daten. Zuerst werden die Daten mit einem gleitendem Mittelwertfilter in der Klasse CMeanFilter geglättet. Anschliessend werden Punkte mit Hilfe des Douglas-Peuker Algorithmuses in der Klasse CSegmentApproximator gelöscht. Sollten es mehrere nicht zusammenhängende Pfade sein müssen diese jetzt noch zusammengesetzt werden.

1.2.4 Roboter Code erstellen

Als letzter Schritt werden die Nutzereinstellungen in die Daten übernommen und der Robotercode erstellt.

Kapitel 2

Hierarchie-Verzeichnis

2.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

CEulerMatrix 9
CGUI 13
CInputParameter 14
CLine3D 26
CLogging 28
CMeanFilter 32
CPathBuilder 44
CPoint3D 46
  CInputPoint3D 22
  COutputPoint3D 37
CRobCodeGenerator 52
CSegmentApproximator 58

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

CEulerMatrix
: Handling und Berechnung Euler Matrix . . . 9
CGUI . . . 13
CInputParameter
: Handling Eingabedaten . . . 14
CInputPoint3D
: Input Punkt . . . 22
CLine3D
: Berechnung Geraden . . . 26
CLogging
: Gleitender Mittelwertfilter . . . 28
CMeanFilter
: Gleitender Mittelwertfilter . . . 32
COutputPoint3D
: Output Punkt . . . 37
CPathBuilder
: Zusammensetzten des Pfades . . . 44
CPoint3D
: Grundklasse Punkt . . . 46
CRobCodeGenerator
: Klasse zum erstellen des Roboter Codes . . . 52
CSegmentApproximator
: Ausdünnen des Pfades . . . 58

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

header/EulerMatrix.h
: Header File handling Euler Matrix . . . 63
header/GUI.h . . . 64
header/InputParameter.h
: Header File Daten Einlesen . . . 64
header/Line3D.h
: Header File Daten Einlesen . . . 65
header/Logging.h
: Logging der Daten . . . 66
header/MeanFilter.h
: Berechnung des gleitenden Mittelwertfilters . . . 67
header/PathBuilder.h
: Setzt die einzelnen Segmente zu einem Vector zusammen . . . 68
header/Point3D.h
: Verarbeitung der Punkte . . . 69
header/RobCodeGenerator.h
: Erstellung des Roboter Codes . . . 70
header/SegmentApproximator.h
: Berechnung des Douglas Peuker Algorithmusses . . . 72
src/EulerMatrix.cpp
: Source Code der Euler Matrix . . . 73
src/GUI.cpp . . . 74
src/InputParameter.cpp
: Source File Daten Einlesen . . . 74
src/Line3D.cpp
: Source File Line3D . . . 76
src/Logging.cpp
: Source File Logging . . . 77
src/MeanFilter.cpp
: Source File gleitender Mittelwertfilter . . . 79
src/PathBuilder.cpp
: Source File Segmente zu Pfad . . . 80
src/Point3D.cpp
: Source File Punkte . . . 81
src/RobCodeGenerator.cpp
: Source File Roboter Code Erstellung . . . 84

src/RobPathEditor.cpp
: Hier wird die main Funktion aufgerufen 86
src/SegmentApproximator.cpp
: Source File Douglas-Peuker 88

Kapitel 5

Klassen-Dokumentation

5.1 CEulerMatrix Klassenreferenz

: Handling und Berechnung Euler Matrix

`#include <EulerMatrix.h>`

fentliche Methoden

- CEulerMatrix (void)
  : Default Konstruktor
- CEulerMatrix (float inputMatrix[3][3])
  : Default Konstruktor
- ~CEulerMatrix ()
  : Dekonstruktor
- void setMatrix (float inputMatrix[3][3])
  : Setzt eine Matrix
- CEulerMatrix getEulerMatrix (void)
  : Auslesen eine Matrix
- void getMatrix (float Matrix[ ][3])
  : Auslesen eine Matrix
- CEulerMatrix angels2mat (double A, double B, double C)
  : Berechnet die neue Umdrehungsmatrix
- tuple< double, double, double > calculateAngels (void)
  : Berechnet die Kuka Wunkel A,B,C

Private Attribute

- float eulerMatrix [3][3]

5.1.1 Ausfhrliche Beschreibung

: Handling und Berechnung Euler Matrix

Diese Klasse speichert die Euler Matrix und hat Funktionen fr Berechnungen mit eben jener.

Definiert in Zeile 18 der Datei EulerMatrix.h.

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 CEulerMatrix() [1/2]

```
CEulerMatrix::CEulerMatrix (
            void  )
```

: Default Konstruktor

Initialisiert die Input Daten mit Null

Siehe auch

: CEulerMatrix(float inputMatrix[3][3])

Definiert in Zeile 10 der Datei EulerMatrix.cpp.

```
00011 {
00012     for (int i = 0; i < 3; i++)
00013     {
00014         for (int m = 0; m < 3; m++)
00015         {
00016             eulerMatrix[i][m] = 0;                  // eulerMatrix mit 0 initialisieren
00017         }
00018     }
00019 }
```

5.1.2.2 CEulerMatrix() [2/2]

```
CEulerMatrix::CEulerMatrix (
            float inputMatrix[3][3] )
```

: Default Konstruktor

Initialisiert die Input Daten mit Null

Parameter

	float inputMatrix[3][3] initialisiert die Klasse mit einer Euler Matrix

Siehe auch

: CEulerMatrix(void)

Definiert in Zeile 21 der Datei EulerMatrix.cpp.

```
00022 {
00023     for (int i = 0; i < 3; i++)
00024     {
00025         for (int m = 0; m < 3; m++)
00026         {
00027             eulerMatrix[i][m] = inputMatrix[i][m];  // eulerMatrix mit Startwerten initialisieren
00028         }
00029     }
00030 }
```

5.1.2.3 ~CEulerMatrix()

```
CEulerMatrix::~CEulerMatrix ( )
```

: Dekonstruktor

Definiert in Zeile 32 der Datei EulerMatrix.cpp.

```
00033 {
00034 }
```

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 angels2mat()

```
CEulerMatrix CEulerMatrix::angels2mat (
            double A,
            double B,
            double C )
```

: Berechnet die neue Umdrehungsmatrix

Parameter

A	double Winkel a
B	double Winkel b
C	double Winkel c

Rckgabe

: float inputMatrix[3][3] gibt die neu berechnete Matrix zurck

Definiert in Zeile 65 der Datei EulerMatrix.cpp.

```
00066 {
00067     float Matrix[3][3];
00068
00069     Matrix[0][0] = cos(A) * cos(C) - sin(A) * cos(B) * sin(C);
00070     Matrix[0][1] = -cos(A) * sin(C) - sin(A) * cos(B) * cos(C);
00071     Matrix[0][2] = sin(A) * sin(B);
00072
00073     Matrix[1][0] = sin(A) * cos(C) + cos(A) * cos(B) * sin(C);
00074     Matrix[1][1] = -sin(A) * sin(C) + cos(A) * cos(B) * cos(C);
00075     Matrix[1][2] = -cos(A) * sin(B);
00076
00077     Matrix[2][0] = sin(B) * sin(C);
00078     Matrix[2][1] = sin(B) * cos(C);
00079     Matrix[2][2] = cos(B);
00080
00081     CEulerMatrix buffer(Matrix);
00082     return buffer;
00083 }
```

5.1.3.2 calculateAngels()

```
tuple< double, double, double > CEulerMatrix::calculateAngels (
            void  )
```

: Berechnet die Kuka Wunkel A,B,C

Rckgabe

: tuple<double , double , double> gibt die berechneten Winkel A, B, C zurück

Definiert in Zeile 86 der Datei EulerMatrix.cpp.

```
00087 {
00088     double a, b, c, sin_a, cos_a, sin_b, abs_cos_b, sin_c, cos_c;
00089
00090     a = atan2(eulerMatrix[1][0], eulerMatrix[0][0]);
00091
00092     sin_a = sin(a);
00093     cos_a = cos(a);
00094     sin_b = eulerMatrix[2][0] * -1;
00095     abs_cos_b = cos(a) * eulerMatrix[0][0] + sin(a) * eulerMatrix[1][0];
00096
00097     b = atan2 (sin_b, abs_cos_b);
00098
00099     sin_c =  sin_a * eulerMatrix[0][2] - cos_a * eulerMatrix[1][2];
00100     cos_c = -sin_a * eulerMatrix[0][1] + cos_a * eulerMatrix[1][1];
00101
00102    c = atan2(sin_c, cos_c);
00103
00104    return make_tuple(a, b, c);
00105 }
```

5.1.3.3 getEulerMatrix()

```
CEulerMatrix CEulerMatrix::getEulerMatrix (
             void  )
```

: Auslesen eine Matrix

Rckgabe

: float inputMatrix[3][3] gibt gespeicherte Matrix zurck

Definiert in Zeile 48 der Datei EulerMatrix.cpp.

```
00049 {
00050     return eulerMatrix;
00051 }
```

5.1.3.4 getMatrix()

```
void CEulerMatrix::getMatrix (
             float Matrix[][3] )
```

: Auslesen eine Matrix

Parameter

	float* inputMatrix[3][3] Pointer zu einer Matrix

Definiert in Zeile 53 der Datei EulerMatrix.cpp.

```
00054 {
00055     for (int i = 0; i < 3; i++)
00056     {
00057         for (int m = 0; m < 3; m++)
00058         {
00059             Matrix[i][m] = eulerMatrix[i][m];  // eulerMatrix mit bergabewerten berschreiben
00060         }
00061     }
00062 }
```

5.1.3.5 setMatrix()

```
void CEulerMatrix::setMatrix (
            float inputMatrix[3][3] )
```

: Setzt eine Matrix

Parameter

	float inputMatrix[3][3] zum setzten einer Matrix

Definiert in Zeile 37 der Datei EulerMatrix.cpp.

```
00038 {
00039     for(int i = 0; i < 3; i++)
00040     {
00041         for (int m = 0; m < 3; m++)
00042         {
00043             eulerMatrix[i][m] = inputMatrix[i][m];  // eulerMatrix mit Übergabewerten überschreiben
00044         }
00045     }
00046 }
```

5.1.4 Dokumentation der Datenelemente

5.1.4.1 eulerMatrix

```
float CEulerMatrix::eulerMatrix[3][3]  [private]
```

Gespeicherte Euler Matrix

Definiert in Zeile 74 der Datei EulerMatrix.h.

Die Dokumentation fr diese Klasse wurde erzeugt aufgrund der Dateien:

- header/EulerMatrix.h
- src/EulerMatrix.cpp

5.2 CGUI Klassenreferenz

```
#include <GUI.h>
```

Öffentliche Methoden

- CGUI ()
- ~CGUI ()

5.2.1 Ausführliche Beschreibung

Definiert in Zeile 3 der Datei GUI.h.

5.2.2 Beschreibung der Konstruktoren und Destruktoren

5.2.2.1 CGUI()

```
CGUI::CGUI ( )
```

Definiert in Zeile 3 der Datei GUI.cpp.
```
00004 {}
```

5.2.2.2 ~CGUI()

```
CGUI::~CGUI ( )
```

Definiert in Zeile 6 der Datei GUI.cpp.
```
00007 {}
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/GUI.h
- src/GUI.cpp

5.3 CInputParameter Klassenreferenz

: Handling Eingabedaten

```
#include <InputParameter.h>
```

Öffentliche Methoden

- CInputParameter (void)
  : Default Konstruktor
- CInputParameter (double initSpeed, bool initSeepManual, bool initOrientationManual, double initA, double initB, double initC)
  : Konstruktor mit Werten
- ~CInputParameter (void)
  : Dekonstruktor
- void setOrientation (bool initOrientationManual, double initA, double initB, double initC)
  : Setzt Orientierungs Daten
- void setSpeed (double initSpeed, bool initSpeedManual)
  : Setzt Geschwindigkeits Daten
- double getSpeed (void)
  : Gibt Geschwindigkeit zurück
- bool getSpeedManual (void)
  : Gibt zurück ob händische Geschwindigkeit verwendet werden soll
- bool getOrientationManual (void)
  : Gibt zurück ob händische Orientierung verwendet werden soll
- tuple< double, double, double > getAngles (void)
  : Gibt Winkel zurück
- void openFile (std::string path)
  : Liest die Daten aus dem Input File ein
- bool detectJump (CInputPoint3D p, double x_prev, double y_prev, double z_prev)
  : Erkennt Sprünge in den Daten
- vector< list< CInputPoint3D > > & getPath ()
  : Gibt Pfad zurück

Private Attribute

- vector< list< CInputPoint3D > > initialPath
- double speed
- bool speedManual
- bool orientationManual
- double A
- double B
- double C
- double difference = 20

5.3.1 Ausführliche Beschreibung

: Handling Eingabedaten

In dieser Klasse werden die eingelesenen einstellbaren Daten und das einlesen der Daten aus der Eingabedatei gehandelt.

Definiert in Zeile 25 der Datei InputParameter.h.

5.3.2 Beschreibung der Konstruktoren und Destruktoren

5.3.2.1 CInputParameter() [1/2]

```
CInputParameter::CInputParameter (
            void  )
```

: Default Konstruktor

Initialisiert die Input Daten mit Null

Siehe auch

: CInputParameter(double initSpeed, bool initSeepManual, bool initOrientationManual, double initA, double initB, double initC)

Initialiserung der Klasse mit 0

Definiert in Zeile 27 der Datei InputParameter.cpp.

```
00028 {
00029     speed = 0;
00030     A = 0;
00031     B = 0;
00032     C = 0;
00033     speedManual = false,
00034     orientationManual = false;
00035
00036 }
```

5.3.2.2 CInputParameter() [2/2]

```
CInputParameter::CInputParameter (
            double initSpeed,
            bool initSpeedManual,
            bool initOrientationManual,
            double initA,
            double initB,
            double initC )
```

: Konstruktor mit Werten

Initialisiert die Input Daten

Parameter

	double initSpeed
	bool initSeepManual
	bool initOrientationManual
	double initA
	double initB
	double initC

Siehe auch

: CInputParameter()

: ∼CInputParameter()

: CInputParameter(void);

Initialiserung der Klasse mit allen Werten

Definiert in Zeile 14 der Datei InputParameter.cpp.

```
00015 {
00016     speed = initSpeed;
00017     speedManual = initSpeedManual;
00018     orientationManual = initOrientationManual;
00019     A = initA;
00020     B = initB;
00021     C = initC;
00022
00023 }
```

5.3.2.3 ∼CInputParameter()

```
CInputParameter::∼CInputParameter (
            void  )
```

: Dekonstruktor

Dekonstruktor

Definiert in Zeile 40 der Datei InputParameter.cpp.

```
00041 {
00042
00043 }
```

5.3.3 Dokumentation der Elementfunktionen

5.3.3.1 detectJump()

```
bool CInputParameter::detectJump (
            CInputPoint3D p,
            double x_prev,
            double y_prev,
            double z_prev )
```

: Erkennt Sprünge in den Daten

Um zu erkennen ob es mehrere Pfade sind wird nach Sprüngen gesucht, bei einem Sprung wird eine neue Liste angefangen.

Parameter

	CInputPoint3D p den aktuellen Punkt
	double x_prev die vorherige x Position
	double y_prev die vorherige y Position
	double z_prev die vorherige z Position

Definiert in Zeile 129 der Datei InputParameter.cpp.

```
00130 {
00131     if(abs(p.getX() - x_prev) > difference)
00132         return true;
00133     else if(abs(p.getY() - y_prev) > difference)
00134         return true;
00135     else if(abs(p.getZ() - z_prev) > difference)
00136         return true;
00137     else
00138         return false;
00139 }
```

5.3.3.2 getAngles()

```
tuple< double, double, double > CInputParameter::getAngles (
             void  )
```

: Gibt Winkel zurück

Gibt die eingegebenen Winkel als tuple zurück

Rückgabe

: tuple <double double double> angles

Definiert in Zeile 79 der Datei InputParameter.cpp.

```
00080 {
00081     return make_tuple(A, B, C);
00082 }
```

5.3.3.3 getOrientationManual()

```
bool CInputParameter::getOrientationManual (
             void  )
```

: Gibt zurück ob händische Orientierung verwendet werden soll

Gibt zurück ob händische Orientierung verwendet werden soll, sonst wird sie später berechnet.

Definiert in Zeile 74 der Datei InputParameter.cpp.

```
00075 {
00076     return orientationManual;
00077 }
```

5.3.3.4 getPath()

```
vector< list< CInputPoint3D > > & CInputParameter::getPath ( )
```

: Gibt Pfad zurück

Rückgabe

: vector<list<CInputPoint3D>> den eingelesen Pfad

Definiert in Zeile 59 der Datei InputParameter.cpp.

```
00060 {
00061     return initialPath;
00062 }
```

5.3.3.5 getSpeed()

```
double CInputParameter::getSpeed (
            void  )
```

: Gibt Geschwindigkeit zurück

Gibt die eingegebene Geschwindigkeit zurück

Definiert in Zeile 64 der Datei InputParameter.cpp.

```
00065 {
00066     return speed;
00067 }
```

5.3.3.6 getSpeedManual()

```
bool CInputParameter::getSpeedManual (
            void  )
```

: Gibt zurück ob händische Geschwindigkeit verwendet werden soll

Gibt zurück ob händische Geschwindigekit verwendet werden soll, sonst wird sie später berechnet.

Definiert in Zeile 69 der Datei InputParameter.cpp.

```
00070 {
00071     return speedManual;
00072 }
```

5.3.3.7 openFile()

```
void CInputParameter::openFile (
            std::string path )
```

: Liest die Daten aus dem Input File ein

Liest die Daten aus einen beliebigen File ein und ruft @detectJump auf um zu erkennen ob es mehrere Aufnahmen sind.

Parameter

	File Pfad

Definiert in Zeile 85 der Datei InputParameter.cpp.

```
00086 {
00087     ifstream fin(path);
00088     char delimiter = ' ';
00089     CInputPoint3D tmpPoint;
00090     CEulerMatrix tmpEuler;
00091     double x, y, z;
00092     double x_prev = 0, y_prev = 0, z_prev = 0;
00093     double timestamp;
00094     int segmentCount = -1;
00095     float dummyMatrix[3][3];
00096
00097
00098     if (!fin.is_open())
00099     {
00100         cerr « "Datei konnte nicht geöffnet werden" « endl;
00101     }
00102     string line;
00103
00104     while(getline(fin, line))
00105     {
00106         std::istringstream sStream (line);
00107         sStream » timestamp » x » y » z » dummyMatrix[0][0] » dummyMatrix[0][1] » dummyMatrix[0][2]
00108             » dummyMatrix[1][0] » dummyMatrix[1][1] » dummyMatrix[1][2] » dummyMatrix[2][0] »
      dummyMatrix[2][1] » dummyMatrix[2][2];
00109
00110         tmpEuler.setMatrix(dummyMatrix);
00111         tmpPoint.setPoint(timestamp, x, y, z, tmpEuler.getEulerMatrix());
00112
00113         if (detectJump(tmpPoint, x_prev, y_prev, z_prev))
00114         {
00115             segmentCount++;
00116             initialPath.push_back(list<CInputPoint3D>());
00117         }
00118
00119         initialPath[segmentCount].push_back(tmpPoint);
00120
00121         x_prev = x;
00122         y_prev = y;
00123         z_prev = z;
00124     }
00125     fin.close();
00126 }
```

5.3.3.8 setOrientation()

```
void CInputParameter::setOrientation (
             bool initOrientationManual,
             double initA,
             double initB,
             double initC )
```

: Setzt Orientierungs Daten

Setzt ob die Orientierung Händisch eingegeben werden soll und die drei Winkel

Parameter

	bool initOrientationManual
	double initA
	double initB
	double initC

Definiert in Zeile 45 der Datei InputParameter.cpp.

```
00046 {
00047     orientationManual = initOrientationManual;
00048     A = initA;
00049     B = initB;
00050     C = initC;
00051 }
```

5.3.3.9 setSpeed()

```
void CInputParameter::setSpeed (
            double initSpeed,
            bool initSpeedManual )
```

: Setzt Geschwindigkeits Daten

Setzt ob die Geschwindigkeit Händisch eingegeben werden soll und die Geschwindigkeit in m/s

Parameter

	double initSpeed
	bool initSeepManual

Definiert in Zeile 53 der Datei InputParameter.cpp.

```
00054 {
00055     speed = initSpeed;
00056     speedManual = initSpeedManual;
00057 }
```

5.3.4 Dokumentation der Datenelemente

5.3.4.1 A

```
double CInputParameter::A  [private]
```

User eingegebener Winkel A

Definiert in Zeile 133 der Datei InputParameter.h.

5.3.4.2 B

```
double CInputParameter::B  [private]
```

User eingegebener Winkel B

Definiert in Zeile 137 der Datei InputParameter.h.

5.3.4.3 C

```
double CInputParameter::C  [private]
```

User eingegebener Winkel C

Definiert in Zeile 141 der Datei InputParameter.h.

5.3.4.4 difference

```
double CInputParameter::difference = 20  [private]
```

Sprung ab dem eine neue Liste angefangen wird

Definiert in Zeile 145 der Datei InputParameter.h.

5.3.4.5 initialPath

```
vector<list<CInputPoint3D> > CInputParameter::initialPath  [private]
```

Vector mit Listen an Input Daten

Definiert in Zeile 117 der Datei InputParameter.h.

5.3.4.6 orientationManual

```
bool CInputParameter::orientationManual  [private]
```

Auswahl ob berechnete oder eingegebene Winkel verwendet werden soll

Definiert in Zeile 129 der Datei InputParameter.h.

5.3.4.7 speed

```
double CInputParameter::speed  [private]
```

User eingegebene Geschwindigkeit

Definiert in Zeile 121 der Datei InputParameter.h.

5.3.4.8 speedManual

```
bool CInputParameter::speedManual  [private]
```

Auswahl ob berechnete oder eingegebene Geschwindigkeit verwendet werden soll

Definiert in Zeile 125 der Datei InputParameter.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

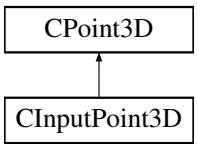
- header/InputParameter.h
- src/InputParameter.cpp

5.4 CInputPoint3D Klassenreferenz

: Input Punkt

`#include <Point3D.h>`

Klassendiagramm für CInputPoint3D:



Öffentliche Methoden

- CInputPoint3D (void)
  : Default Konstruktor
- CInputPoint3D (double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)
  : Default Konstruktor
- ~CInputPoint3D (void)
  : Dekonstruktor
- double getTime ()
  : Gibt den Zeitstempel zurück
- CEulerMatrix getEulerMatrix ()
  : Gibt die gespeicherte Eulermatrix zurück
- void setTime (double time)
  : Setzt den Zeitstempel
- void setEulerMatrix (CEulerMatrix orientation)
  : Setzt die Eulermatrix
- void setPoint (double time, double X, double Y, double Z, CEulerMatrix orientation)
  : Setzt einen Input Punkt

Öffentliche Methoden geerbt von CPoint3D

- CPoint3D (void)
  : Default Konstruktor
- CPoint3D (double X, double Y, double Z)
  : Default Konstruktor
- ~CPoint3D (void)
  : Dekonstruktor
- double getX ()
  : Gibt X zurück
- double getY ()
  : Gibt Y zurück
- double getZ ()
  : Gibt Z zurück
- void setX (double X)
  : Setzt X
- void setY (double Y)

: Setzt Y

- void setZ (double Z)

  : Setzt Z
- void set (double X, double Y, double Z)

  : Setzt X, Y und Z
- double distanceTo (CPoint3D point)

  : Berechnet die Distanz zu einem anderen Punkt
- double distanceTo (CLine3D line)

  : Berechnet die Distanz zu einer Linie

Private Attribute

- double timestamp
- CEulerMatrix orientationMatrix

Weitere Geerbte Elemente

Geschützte Attribute geerbt von CPoint3D

- double x
- double y
- double z

5.4.1 Ausführliche Beschreibung

: Input Punkt

Kind der Punkt Grundklasse, erweitert um den Timestamp und die Eulermatrix

Definiert in Zeile 106 der Datei Point3D.h.

5.4.2 Beschreibung der Konstruktoren und Destruktoren

5.4.2.1 CInputPoint3D() [1/2]

```
CInputPoint3D::CInputPoint3D (
            void  )
```

: Default Konstruktor

Initialisiert des eingelesenen Punktes mit Null

Siehe auch

: CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)

Definiert in Zeile 107 der Datei Point3D.cpp.

```
00107                                   : CPoint3D()
00108 {
00109     timestamp = 0;
00110 }
```

5.4.2.2 CInputPoint3D() [2/2]

```
CInputPoint3D::CInputPoint3D (
            double X,
            double Y,
            double Z,
            double Timestamp,
            CEulerMatrix Matrix )
```

: Default Konstruktor

Initialisiert des eingelesenen Punktes mit Null

Parameter

	double X
	double Y
	double Z
	double Timestamp
	CEulerMatrix Matrix

Siehe auch

: CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)

Definiert in Zeile 112 der Datei Point3D.cpp.

```
00113 {
00114     x = X;
00115     y = Y;
00116     z = Z;
00117     timestamp = Timestamp;
00118     orientationMatrix = Matrix;
00119
00120 }
```

5.4.2.3 ~CInputPoint3D()

```
CInputPoint3D::~CInputPoint3D (
            void  )
```

: Dekonstruktor

Definiert in Zeile 122 der Datei Point3D.cpp.

```
00123 {
00124 }
```

5.4.3 Dokumentation der Elementfunktionen

5.4.3.1 getEulerMatrix()

```
CEulerMatrix CInputPoint3D::getEulerMatrix ( )
```

: Gibt die gespeicherte Eulermatrix zurück

Rückgabe

: CEulerMatrix

Definiert in Zeile 144 der Datei Point3D.cpp.

```
00145 {
00146     return orientationMatrix;
00147 }
```

5.4.3.2 getTime()

```
double CInputPoint3D::getTime ( )
```

: Gibt den Zeitstempel zurück

Rückgabe

: double Zeitstempel

Definiert in Zeile 149 der Datei Point3D.cpp.

```
00150 {
00151     return timestamp;
00152 }
```

5.4.3.3 setEulerMatrix()

```
void CInputPoint3D::setEulerMatrix (
            CEulerMatrix orientation )
```

: Setzt die Eulermatrix

Parameter

	CEulerMatrix orientation

Definiert in Zeile 126 der Datei Point3D.cpp.

```
00127 {
00128     orientationMatrix = orientation;
00129 }
```

5.4.3.4 setPoint()

```
void CInputPoint3D::setPoint (
            double time,
            double X,
            double Y,
            double Z,
            CEulerMatrix orientation )
```

: Setzt einen Input Punkt

Parameter

	double time
	double X
	double Y
	double Z
	CEulerMatrix orientation

Definiert in Zeile 132 der Datei Point3D.cpp.

```
00133 {
```

```
00134     setTime(time);
00135     set(X, Y, Z);
00136     setEulerMatrix(orientation);
00137 }
```

5.4.3.5 setTime()

```
void CInputPoint3D::setTime (
            double time )
```

: Setzt den Zeitstempel

Parameter

	double time

Definiert in Zeile 139 der Datei Point3D.cpp.

```
00140 {
00141     timestamp = time;
00142 }
```

5.4.4 Dokumentation der Datenelemente

5.4.4.1 orientationMatrix

```
CEulerMatrix CInputPoint3D::orientationMatrix  [private]
```

Eulermatrix des Punktes

Definiert in Zeile 170 der Datei Point3D.h.

5.4.4.2 timestamp

```
double CInputPoint3D::timestamp  [private]
```

Zeitstempel

Definiert in Zeile 166 der Datei Point3D.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/Point3D.h
- src/Point3D.cpp

5.5 CLine3D Klassenreferenz

: Berechnung Geraden

```
#include <Line3D.h>
```

Öffentliche Methoden

- CLine3D (void)
  : Default Konstruktor
- CLine3D (CPoint3D P1, CPoint3D P2)
  : Konstruktor mit zwei Punkten
- ~CLine3D (void)
  : Dekonstruktor

Öffentliche Attribute

- CPoint3D p1
- CPoint3D p2

5.5.1 Ausführliche Beschreibung

: Berechnung Geraden

In dieser Klasse werden alle Berechnungen die zwischen zwei Punken passieren gehandhabt.

Definiert in Zeile 18 der Datei Line3D.h.

5.5.2 Beschreibung der Konstruktoren und Destruktoren

5.5.2.1 CLine3D() [1/2]

```
CLine3D::CLine3D (
            void  )
```

: Default Konstruktor

Initialisiert die Klasse

Siehe auch

: CLine3D(CPoint3D P1, CPoint3D P2)

Definiert in Zeile 10 der Datei Line3D.cpp.

```
00011 {
00012 }
```

5.5.2.2 CLine3D() [2/2]

```
CLine3D::CLine3D (
            CPoint3D P1,
            CPoint3D P2 )
```

: Konstruktor mit zwei Punkten

Initialisiert die Klasse

Siehe auch

: CLine3D(void);

Definiert in Zeile 14 der Datei Line3D.cpp.

```
00015 {
00016     p1 = P1;
00017     p2 = P2;
00018 }
```

5.5.2.3 ~CLine3D()

```
CLine3D::~CLine3D (
            void  )
```

: Dekonstruktor

Definiert in Zeile 20 der Datei Line3D.cpp.

```
00021 {
00022 }
```

5.5.3 Dokumentation der Datenelemente

5.5.3.1 p1

```
CPoint3D CLine3D::p1
```

Punkt 1

Definiert in Zeile 41 der Datei Line3D.h.

5.5.3.2 p2

```
CPoint3D CLine3D::p2
```

Punkt 2

Definiert in Zeile 45 der Datei Line3D.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/Line3D.h
- src/Line3D.cpp

5.6 CLogging Klassenreferenz

: Gleitender Mittelwertfilter

```
#include <Logging.h>
```

Öffentliche Methoden

- CLogging (void)
  : Default Konstruktor
- CLogging (string path)
  : Default Konstruktor
- ~CLogging (void)
  : Dekonstruktor
- void setStep (int Step)
  : Setzt in welchem Schritt wir uns befinden
- void logData (vector< list< CInputPoint3D > > &sourcePath)
  : Ruft calculateMean für die einzelnen Segmente auf
- void logData (vector< CInputPoint3D > &sourcePath)
  : Ruft calculateMean für die einzelnen Segmente auf

Private Attribute

- int step
- string path

5.6.1 Ausführliche Beschreibung

: Gleitender Mittelwertfilter

In dieser Klasse werden die eingelesenen Daten mit einem gleitenden Mittelwertfilter mit einstellbarem Fenster geglättet.

Definiert in Zeile 22 der Datei Logging.h.

5.6.2 Beschreibung der Konstruktoren und Destruktoren

5.6.2.1 CLogging() [1/2]

```
CLogging::CLogging (
            void  )
```

: Default Konstruktor

Initialisiert Logging Klasse

Siehe auch

: CMeanFilter(int Window);

Definiert in Zeile 9 der Datei Logging.cpp.

```
00010 {
00011     step = 0;
00012 }
```

5.6.2.2 CLogging() [2/2]

```
CLogging::CLogging (
            string path )
```

: Default Konstruktor

Initialisiert Logging Klasse

Siehe auch

: CMeanFilter(int Window);

Definiert in Zeile 14 der Datei Logging.cpp.

```
00015 {
00016     path = Path;
00017 }
```

5.6.2.3 ~CLogging()

```
CLogging::~CLogging (
            void  )
```

: Dekonstruktor

Definiert in Zeile 19 der Datei Logging.cpp.

```
00020 {
00021
00022 }
```

5.6.3 Dokumentation der Elementfunktionen

5.6.3.1 logData() [1/2]

```
void CLogging::logData (
            vector< CInputPoint3D > & sourcePath )
```

: Ruft calculateMean für die einzelnen Segmente auf

Loggingdaten werden weggespeichert

Parameter

	vector<CInputPoint3D>& sourcePath

Definiert in Zeile 84 der Datei Logging.cpp.

```
00085 {
00086     string filepath;
00087     float dummyMatrix[3][3];
00088     CEulerMatrix tmpEuler;
00089
00090     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00091
00092     FILE* fid = fopen(filepath.c_str(), "w");
00093
00094     if (fid == NULL)
00095     {
00096         cerr « "ERROR - Can NOT write to output file!\n";
00097         return;
00098     }
00099
00100     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00101         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00102         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00103
00104     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00105     {
00106         tmpEuler.getMatrix(dummyMatrix);
00107
00108         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)sourcePath[s].getTime(),
00109             (double)sourcePath[s].getX(), (double)sourcePath[s].getY(), (double)sourcePath[s].getZ(),
00110             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00111             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00112             dummyMatrix[2][2]);
00113     }
00114
00115     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00116         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00117         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00118 }
```

5.6.3.2 logData() [2/2]

```
void CLogging::logData (
```

```
vector< list< CInputPoint3D > > & sourcePath )
```

: Ruft calculateMean für die einzelnen Segmente auf

Loggingdaten werden weggespeichert

Parameter

	vector<list<CInputPoint3D>>& sourcePath

Definiert in Zeile 30 der Datei Logging.cpp.

```
00031 {
00032     string filepath;
00033     float dummyMatrix[3][3];
00034     CEulerMatrix tmpEuler;
00035
00036     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00037
00038     FILE* fid = fopen(filepath.c_str(), "w");
00039
00040     if (fid == NULL)
00041     {
00042         cerr « "ERROR - Can NOT write to output file!\n";
00043         return;
00044     }
00045
00046     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f\n",
00047         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00048         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00049
00050     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00051     {
00052         list<CInputPoint3D>::iterator itr = sourcePath[s].begin();
00053
00054         tmpEuler = itr->getEulerMatrix();
00055         tmpEuler.getMatrix(dummyMatrix);
00056
00057         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00058             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00059             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00060             dummyMatrix[2][2]);
00061
00062
00063         for (; itr != sourcePath[s].end(); itr++) //for all points in the segment
00064         {
00065             fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00066                 dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00067                 dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00068                 dummyMatrix[2][2]);
00069         }
00070
00071         itr--;
00072         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00073             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00074             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00075             dummyMatrix[2][2]);
00076     }
00077
00078     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00079         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00080         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00081 }
```

5.6.3.3 setStep()

```
void CLogging::setStep (
            int Step )
```

: Setzt in welchem Schritt wir uns befinden

Parameter

	int Step

Definiert in Zeile 24 der Datei Logging.cpp.

```
00025 {
00026     step = Step;
00027 }
```

5.6.4 Dokumentation der Datenelemente

5.6.4.1 path

```
string CLogging::path  [private]
```

Speicherpfad

Definiert in Zeile 66 der Datei Logging.h.

5.6.4.2 step

```
int CLogging::step  [private]
```

In welchem Schritt sind wir gerade

Definiert in Zeile 62 der Datei Logging.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/Logging.h
- src/Logging.cpp

5.7 CMeanFilter Klassenreferenz

: Gleitender Mittelwertfilter

```
#include <MeanFilter.h>
```

Öffentliche Methoden

- CMeanFilter ()
  : Default Konstruktor
- CMeanFilter (int Window)
  : Konstruktor
- ~CMeanFilter ()
  : Dekonstruktor
- void setWindowSize (int Window)
  : Setzt das Fenster
- int getWindowSize ()
  : Gibt das gesetzte Fenster zurück
- vector< list< CInputPoint3D > > & getPath ()
  : Gibt den geglätteten Pfad zurück
- list< CInputPoint3D > calculateMean (list< CInputPoint3D > &segment)
  : Berechnet gleitenden Mittelwert
- void mean (vector< list< CInputPoint3D > > &sourcePath, CLogging log)
  : Ruft calculateMean für die einzelnen Segmente auf

Private Attribute

- int windowSize
- vector< list< CInputPoint3D > > meanPath

5.7.1 Ausführliche Beschreibung

: Gleitender Mittelwertfilter

In dieser Klasse werden die eingelesenen Daten mit einem gleitenden Mittelwertfilter mit einstellbarem Fenster geglättet.

Definiert in Zeile 21 der Datei MeanFilter.h.

5.7.2 Beschreibung der Konstruktoren und Destruktoren

5.7.2.1 CMeanFilter() [1/2]

```
CMeanFilter::CMeanFilter ( )
```

: Default Konstruktor

Initialisiert des Fensters mit 3 als default Wert

Siehe auch

: CMeanFilter(int Window);

Definiert in Zeile 11 der Datei MeanFilter.cpp.

```
00012 {
00013     windowSize = 3;
00014 }
```

5.7.2.2 CMeanFilter() [2/2]

```
CMeanFilter::CMeanFilter (
            int Window )
```

: Konstruktor

Initialisiert die Input Daten mit Null

Parameter

	int Window Konstruktor der Klasse mit Fenster

Siehe auch

:CMeanFilter();

Definiert in Zeile 16 der Datei MeanFilter.cpp.

```
00017 {
00018     windowSize = Window;
00019 }
```

5.7.2.3 ~CMeanFilter()

```
CMeanFilter::~CMeanFilter ( )
```

: Dekonstruktor

Definiert in Zeile 21 der Datei MeanFilter.cpp.

```
00022 {
00023 }
```

5.7.3 Dokumentation der Elementfunktionen

5.7.3.1 calculateMean()

```
list< CInputPoint3D > CMeanFilter::calculateMean (
            list< CInputPoint3D > & segment )
```

: Berechnet gleitenden Mittelwert

Hier wird der Mittelwert der einzelnen Segmente berechnet

Parameter

	list<CInputPoint3D>& segment bekommt einzelne Segmente

Rückgabe

: list<CInputPoint3D> gibt gelättete Segmente zurück

Definiert in Zeile 52 der Datei MeanFilter.cpp.

```
00053 {
00054     double sumX = 0, sumY = 0, sumZ = 0;
00055     double div = 0;
00056     int m = 0;
00057     int OffsetPos = 0;
00058     int OffsetNeg = 0;
00059
00060     CInputPoint3D p;
00061
00062     size_t inputSize = segment.size();
00063
00064     list<CInputPoint3D>::iterator it = segment.begin();
00065     list<CInputPoint3D> newSegment;
00066
00067     for (size_t i = 0; i < inputSize - windowSize; ++i)
00068     {
00069         sumX = 0, sumY = 0, sumZ = 0;
00070         div = 0;
00071         p.setTime(it->getTime());
00072         p.setEulerMatrix(it->getEulerMatrix());
00073         for (size_t j = i; j < i + windowSize; ++j)
00074         {
00075
00076             sumX += it->getX();
00077             sumY += it->getY();
00078             sumZ += it->getZ();
00079             div++;
```

```
00080             it++;
00081         }
00082         for (size_t index = windowSize; index > 0; index--)
00083         {
00084             it--;
00085         }
00086         p.set(sumX / div, sumY / div, sumZ / div);
00087         if(it != segment.end())
00088             it++;
00089         newSegment.push_back(p);
00090     }
00091     return newSegment;
00092 }
```

5.7.3.2 getPath()

```
vector< list< CInputPoint3D > > & CMeanFilter::getPath ( )
```

: Gibt den geglätteten Pfad zurück

Rückgabe

: vector<list<CInputPoint3D>>

Definiert in Zeile 35 der Datei MeanFilter.cpp.

```
00036 {
00037     return meanPath;
00038 }
```

5.7.3.3 getWindowSize()

```
int CMeanFilter::getWindowSize ( )
```

: Gibt das gesetzte Fenster zurück

Rückgabe

: Window int

Definiert in Zeile 30 der Datei MeanFilter.cpp.

```
00031 {
00032     return windowSize;
00033 }
```

5.7.3.4 mean()

```
void CMeanFilter::mean (
            vector< list< CInputPoint3D > > & sourcePath,
            CLogging log )
```

: Ruft calculateMean für die einzelnen Segmente auf

Hier wird durch die Segmente interiert und für jedes die calculate Mean Funktion aufgerufen. Anschliessend werden sie in meanPath abgespeichert.

Parameter

	list<CInputPoint3D>& segment bekommt einzelne Segmente
	CLogging log für das Logging

Definiert in Zeile 40 der Datei MeanFilter.cpp.

```
00041 {
00042     list<CInputPoint3D> dummyList;
00043     for (size_t s = 0; s < sourcePath.size(); s++)
00044     {
00045         dummyList = calculateMean(sourcePath[s]);
00046         meanPath.push_back(dummyList);
00047     }
00048     log.setStep(2);
00049     log.logData(meanPath);
00050 }
```

5.7.3.5 setWindowSize()

```
void CMeanFilter::setWindowSize (
             int Window )
```

: Setzt das Fenster

Parameter

	Window int

Definiert in Zeile 25 der Datei MeanFilter.cpp.

```
00026 {
00027     windowSize = Window;
00028 }
```

5.7.4 Dokumentation der Datenelemente

5.7.4.1 meanPath

```
vector<list<CInputPoint3D> > CMeanFilter::meanPath  [private]
```

Hier werden die geglätteten Daten gespeichert

Definiert in Zeile 83 der Datei MeanFilter.h.

5.7.4.2 windowSize

```
int CMeanFilter::windowSize  [private]
```

Grösse des Fensters des gleitenden Mittelwerts

Definiert in Zeile 79 der Datei MeanFilter.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

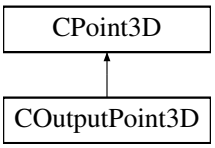
- header/MeanFilter.h
- src/MeanFilter.cpp

5.8 COutputPoint3D Klassenreferenz

: Output Punkt

`#include <Point3D.h>`

Klassendiagramm für COutputPoint3D:



Öffentliche Methoden

- COutputPoint3D (void)
  : Default Konstruktor
- COutputPoint3D (double Speed, double X, double Y, double Z, double A, double B, double C)
  : Default Konstruktor
- ~COutputPoint3D (void)
  : Dekonstruktor
- double getSpeed ()
  : Gibt die Geschwindigkeit zurück
- double getA ()
  : Gibt A zurück
- double getB ()
  : Gibt B zurück
- double getC ()
  : Gibt C zurück
- void setSpeed (double speed)
  : Setzt die Geschwindigkeit
- void setA (double A)
  : Setzt A
- void setB (double B)
  : Setzt B
- void setC (double C)
  : Setzt C

Öffentliche Methoden geerbt von CPoint3D

- CPoint3D (void)
  : Default Konstruktor
- CPoint3D (double X, double Y, double Z)
  : Default Konstruktor
- ~CPoint3D (void)
  : Dekonstruktor
- double getX ()
  : Gibt X zurück
- double getY ()

: Gibt Y zurück

- double getZ ()

  : Gibt Z zurück
- void setX (double X)

  : Setzt X
- void setY (double Y)

  : Setzt Y
- void setZ (double Z)

  : Setzt Z
- void set (double X, double Y, double Z)

  : Setzt X, Y und Z
- double distanceTo (CPoint3D point)

  : Berechnet die Distanz zu einem anderen Punkt
- double distanceTo (CLine3D line)

  : Berechnet die Distanz zu einer Linie

Private Attribute

- double a
- double b
- double c
- double speed

Weitere Geerbte Elemente

Geschützte Attribute geerbt von CPoint3D

- double x
- double y
- double z

5.8.1 Ausführliche Beschreibung

: Output Punkt

Kind der Punkt Grundklasse, erweitert um die Geschwindigkeit und die Drehwinkel

Definiert in Zeile 177 der Datei Point3D.h.

5.8.2 Beschreibung der Konstruktoren und Destruktoren

5.8.2.1 COutputPoint3D() [1/2]

```
COutputPoint3D::COutputPoint3D (
            void  )
```

: Default Konstruktor

Initialisiert des fertig bearbeiteten Punktes mit Null

Siehe auch

: COutputPoint3D(double Speed, double X, double Y, double Z, double A, double B, double C);

Definiert in Zeile 156 der Datei Point3D.cpp.

```
00156                                   : CPoint3D()
00157 {
00158     speed = 0;
00159     a = 0;
00160     b = 0;
00161     c = 0;
00162 }
```

5.8.2.2 COutputPoint3D() [2/2]

```
COutputPoint3D::COutputPoint3D (
            double Speed,
            double X,
            double Y,
            double Z,
            double A,
            double B,
            double C )
```

: Default Konstruktor

Initialisiert des eingelesenen Punktes mit Null

Parameter

	double Speed
	double X
	double Y
	double Z
	double A
	double B
	double C

Siehe auch

: CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)

Definiert in Zeile 164 der Datei Point3D.cpp.

```
00165 {
00166     speed = Speed;
00167     a = A;
00168     b = B;
00169     c = C;
00170     x = X;
00171     y = Y;
00172     z = Z;
00173 }
```

5.8.2.3 ~COutputPoint3D()

```
COutputPoint3D::~COutputPoint3D (
            void  )
```

: Dekonstruktor

Definiert in Zeile 175 der Datei Point3D.cpp.

```
00176 {
00177
00178 }
```

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 getA()

```
double COutputPoint3D::getA (
            void  )
```

: Gibt A zurück

Rückgabe

: double A

Definiert in Zeile 180 der Datei Point3D.cpp.

```
00181 {
00182     return a;
00183 }
```

5.8.3.2 getB()

```
double COutputPoint3D::getB (
            void  )
```

: Gibt B zurück

Rückgabe

: double B

Definiert in Zeile 185 der Datei Point3D.cpp.

```
00186 {
00187     return b;
00188 }
```

5.8.3.3 getC()

```
double COutputPoint3D::getC (
            void  )
```

: Gibt C zurück

Rückgabe

: double C

Definiert in Zeile 190 der Datei Point3D.cpp.

```
00191 {
00192     return c;
00193 }
```

5.8.3.4 getSpeed()

```
double COutputPoint3D::getSpeed (
            void  )
```

: Gibt die Geschwindigkeit zurück

Rückgabe

: double Geschwindigkeit

Definiert in Zeile 195 der Datei Point3D.cpp.

```
00196 {
00197     return speed;
00198 }
```

5.8.3.5 setA()

```
void COutputPoint3D::setA (
            double A )
```

: Setzt A

Parameter

	double A

Definiert in Zeile 200 der Datei Point3D.cpp.

```
00201 {
00202     a = A;
00203 }
```

5.8.3.6 setB()

```
void COutputPoint3D::setB (
            double B )
```

: Setzt B

Parameter

	double B

Definiert in Zeile 205 der Datei Point3D.cpp.

```
00206 {
00207     b = B;
00208 }
```

5.8.3.7 setC()

```
void COutputPoint3D::setC (
            double C )
```

: Setzt C

Parameter

	double C

Definiert in Zeile 210 der Datei Point3D.cpp.

```
00211 {
00212     c = C;
00213 }
```

5.8.3.8 setSpeed()

```
void COutputPoint3D::setSpeed (
            double speed )
```

: Setzt die Geschwindigkeit

Parameter

	double speed

Definiert in Zeile 215 der Datei Point3D.cpp.

```
00216 {
00217     speed = Speed;
00218 }
```

5.8.4 Dokumentation der Datenelemente

5.8.4.1 a

```
double COutputPoint3D::a  [private]
```

Drehwinkel des Punktes

Definiert in Zeile 249 der Datei Point3D.h.

5.8.4.2 b

```
double COutputPoint3D::b  [private]
```

Definiert in Zeile 249 der Datei Point3D.h.

5.8.4.3 c

```
double COutputPoint3D::c  [private]
```

Definiert in Zeile 249 der Datei Point3D.h.

5.8.4.4 speed

```
double COutputPoint3D::speed  [private]
```

Geschwindigkeit zum Punkt hin? TODO: überprüfen

Definiert in Zeile 253 der Datei Point3D.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/Point3D.h
- src/Point3D.cpp

5.9 CPathBuilder Klassenreferenz

: Zusammensetzten des Pfades

```
#include <PathBuilder.h>
```

Öffentliche Methoden

- CPathBuilder (void)
  : Default Konstruktor
- ~CPathBuilder (void)
  : Dekonstruktor
- vector< CInputPoint3D > & getPath ()
  : Gibt Pfad zurück
- void createPath (vector< list< CInputPoint3D > > &segments, CLogging log)
  : Gibt Pfad zurück

Private Attribute

- vector< CInputPoint3D > path

5.9.1 Ausführliche Beschreibung

: Zusammensetzten des Pfades

In dieser Klasse wird aus den Segmenten ein zusammenhängender Pfad erstellt

Definiert in Zeile 21 der Datei PathBuilder.h.

5.9.2 Beschreibung der Konstruktoren und Destruktoren

5.9.2.1 CPathBuilder()

```
CPathBuilder::CPathBuilder (
            void  )
```

: Default Konstruktor

Initialisiert der Klasse

Definiert in Zeile 9 der Datei PathBuilder.cpp.

```
00010 {
00011 }
```

5.9.2.2 ~CPathBuilder()

```
CPathBuilder::~CPathBuilder (
            void  )
```

: Dekonstruktor

Definiert in Zeile 14 der Datei PathBuilder.cpp.

```
00015 {
00016 }
```

5.9.3 Dokumentation der Elementfunktionen

5.9.3.1 createPath()

```
void CPathBuilder::createPath (
            vector< list< CInputPoint3D > > & segments,
            CLogging log )
```

: Gibt Pfad zurück

Parameter

	segments vector<list<CInputPoint3D>>& Pfad aus Segmenten
	CLogging log für das Logging

Definiert in Zeile 23 der Datei PathBuilder.cpp.

```
00024 {
00025     CInputPoint3D point; //startpoint
00026     path.push_back(point);
00027
00028     for (size_t s = 0; s < segments.size(); s++) //for all segments
00029     {
00030         list<CInputPoint3D>::iterator itr = segments[s].begin();
00031
00032         point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ()); //point over start
      of segment
00033         path.push_back(point);
00034
00035         for (; itr != segments[s].end(); itr++) //for all points in the segment
00036         {
00037             point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ());
```

```
00038             path.push_back(point);
00039         }
00040
00041         itr--;
00042
00043         point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ()); //point over end of
      segment
00044         path.push_back(point);
00045     }
00046
00047     point.set(0, 0, 0); //endpoint (== startpoint)
00048     path.push_back(point);
00049
00050     log.setStep(4);
00051     log.logData(path);
00052 }
```

5.9.3.2 getPath()

```
vector< CInputPoint3D > & CPathBuilder::getPath ( )
```

: Gibt Pfad zurück

Rückgabe

: vector<CInputPoint3D> zusammengesetzter Pfad

Definiert in Zeile 18 der Datei PathBuilder.cpp.

```
00019 {
00020     return path;
00021 }
```

5.9.4 Dokumentation der Datenelemente

5.9.4.1 path

```
vector<CInputPoint3D> CPathBuilder::path  [private]
```

Vector mit den zusammengesetzten Pfad Daten

Definiert in Zeile 50 der Datei PathBuilder.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

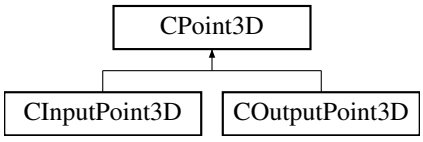
- header/PathBuilder.h
- src/PathBuilder.cpp

5.10 CPoint3D Klassenreferenz

: Grundklasse Punkt

```
#include <Point3D.h>
```

Klassendiagramm für CPoint3D:



Öffentliche Methoden

- CPoint3D (void)
  : Default Konstruktor
- CPoint3D (double X, double Y, double Z)
  : Default Konstruktor
- ~CPoint3D (void)
  : Dekonstruktor
- double getX ()
  : Gibt X zurück
- double getY ()
  : Gibt Y zurück
- double getZ ()
  : Gibt Z zurück
- void setX (double X)
  : Setzt X
- void setY (double Y)
  : Setzt Y
- void setZ (double Z)
  : Setzt Z
- void set (double X, double Y, double Z)
  : Setzt X, Y und Z
- double distanceTo (CPoint3D point)
  : Berechnet die Distanz zu einem anderen Punkt
- double distanceTo (CLine3D line)
  : Berechnet die Distanz zu einer Linie

Geschützte Attribute

- double x
- double y
- double z

5.10.1 Ausführliche Beschreibung

: Grundklasse Punkt

Das ist die Grundklasse eines Punktes, hier lassen sich die Basiswerte setzten und Abstände zwishen Punkten berechnen.

Definiert in Zeile 20 der Datei Point3D.h.

5.10.2 Beschreibung der Konstruktoren und Destruktoren

5.10.2.1 CPoint3D() [1/2]

```
CPoint3D::CPoint3D (
            void  )
```

: Default Konstruktor

Initialisiert des Punktes mit Null

Siehe auch

: CPoint3D(double X, double Y, double Z)

Definiert in Zeile 13 der Datei Point3D.cpp.

```
00014 {
00015     x = 0;
00016     y = 0;
00017     z = 0;
00018 }
```

5.10.2.2 CPoint3D() [2/2]

```
CPoint3D::CPoint3D (
            double X,
            double Y,
            double Z )
```

: Default Konstruktor

Initialisiert des Punktes mit Null

Parameter

	double X
	double Y
	double Z

Siehe auch

: CPoint3D(void)

Definiert in Zeile 20 der Datei Point3D.cpp.

```
00021 {
00022     x = X;
00023     y = Y;
00024     z = Z;
00025 }
```

5.10.2.3 ~CPoint3D()

```
CPoint3D::~CPoint3D (
            void  )
```

: Dekonstruktor

Definiert in Zeile 27 der Datei Point3D.cpp.

```
00028 {
00029 }
```

5.10.3 Dokumentation der Elementfunktionen

5.10.3.1 distanceTo() [1/2]

```
double CPoint3D::distanceTo (
            CLine3D line )
```

: Berechnet die Distanz zu einer Linie

Parameter

	CLine3D line

Rückgabe

: double Distanz

Definiert in Zeile 73 der Datei Point3D.cpp.

```
00074 {
00075     double bx, by, bz, rv_sq, dist, vp1, vp2, vp3;          // Variablen Anlegen
00076
00077     /*
00078     Vermessen wird der Punkt selbst
00079
00080     bx, by, bz      == Vektordifferenz
00081     rv_sq           == Betrag des Linienvektors
00082     dist            == Distanz von Punkt zu Linie
00083     vp1, vp2, vp3   == Vektorprodukte
00084     */
00085
00086     int rvx = line.p1.x - line.p2.x;      // Parameter X des Linienvektor berechnen
00087     int rvy = line.p1.y - line.p2.y;      // Parameter Y des Linienvektor berechnen
00088     int rvz = line.p1.z - line.p2.z;      // Parameter Z des Linienvektor berechnen
00089
00090     rv_sq = sqrt(((double)rvx * (double)rvx) + ((double)rvy * (double)rvy) + ((double)rvz *
      (double)rvz));      // Betrag des Linienvektor berechnen
00091
00092     bx = x - (double)line.p1.x;             // X(Punkt) - X(Aufpunkt)
00093     by = y - (double)line.p1.y;             // Y(Punkt) - Y(Aufpunkt)
00094     bz = z - (double)line.p1.z;             // Z(Punkt) - Z(Aufpunkt)
00095
00096     vp1 = by * rvz - bz * rvy;              // Parameter X Vektorprodukt
00097     vp2 = bz * rvx - bx * rvz;              // Parameter Y Vektorprodukt
00098     vp3 = bx * rvy - by * rvx;              // Parameter Z Vektorprodukt
00099
00100     dist = sqrt(vp1 * vp1 + vp2 * vp2 + vp3 * vp3) / rv_sq; // Betrag des Vektors berechnen
00101
00102     return dist;
00103 }
```

5.10.3.2 distanceTo() [2/2]

```
double CPoint3D::distanceTo (
            CPoint3D point )
```

: Berechnet die Distanz zu einem anderen Punkt

Parameter

	CPoint3D point

Rückgabe

: double Distanz

Definiert in Zeile 68 der Datei Point3D.cpp.

```
00069 {
00070     return sqrt(pow((double)(x - (double)point.getX()), 2) + pow((double)(y - (double)point.getY()),
      2) + pow((double)(z - (double)point.getZ()), 2));   // Pythagoras 3D
00071 }
```

5.10.3.3 getX()

```
double CPoint3D::getX (
            void  )
```

: Gibt X zurück

Rückgabe

: double

Definiert in Zeile 31 der Datei Point3D.cpp.

```
00032 {
00033     return x;
00034 }
```

5.10.3.4 getY()

```
double CPoint3D::getY (
            void  )
```

: Gibt Y zurück

Rückgabe

: double

Definiert in Zeile 36 der Datei Point3D.cpp.

```
00037 {
00038     return y;
00039 }
```

5.10.3.5 getZ()

```
double CPoint3D::getZ (
            void  )
```

: Gibt Z zurück

Rückgabe

: double

Definiert in Zeile 41 der Datei Point3D.cpp.

```
00042 {
00043     return z;
00044 }
```

5.10.3.6 set()

```
void CPoint3D::set (
            double X,
            double Y,
            double Z )
```

: Setzt X, Y und Z

Parameter

	double X
	double Y
	double Z

Definiert in Zeile 61 der Datei Point3D.cpp.

```
00062 {
00063     x = X;
00064     y = Y;
00065     z = Z;
00066 }
```

5.10.3.7 setX()

```
void CPoint3D::setX (
            double X )
```

: Setzt X

Parameter

	double X

Definiert in Zeile 46 der Datei Point3D.cpp.

```
00047 {
00048     x = X;
00049 }
```

5.10.3.8 setY()

```
void CPoint3D::setY (
            double Y )
```

: Setzt Y

Parameter

	double Y

Definiert in Zeile 51 der Datei Point3D.cpp.

```
00052 {
00053     y = Y;
00054 }
```

5.10.3.9 setZ()

```
void CPoint3D::setZ (
            double Z )
```

: Setzt Z

Parameter

	double Z

Definiert in Zeile 56 der Datei Point3D.cpp.

```
00057 {
00058     z = Z;
00059 }
```

5.10.4 Dokumentation der Datenelemente

5.10.4.1 x

```
double CPoint3D::x  [protected]
```

Koordinaten des Punkts

Definiert in Zeile 99 der Datei Point3D.h.

5.10.4.2 y

```
double CPoint3D::y  [protected]
```

Definiert in Zeile 99 der Datei Point3D.h.

5.10.4.3 z

```
double CPoint3D::z  [protected]
```

Definiert in Zeile 99 der Datei Point3D.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/Point3D.h
- src/Point3D.cpp

5.11 CRobCodeGenerator Klassenreferenz

: Klasse zum erstellen des Roboter Codes

```
#include <RobCodeGenerator.h>
```

Öffentliche Methoden

- CRobCodeGenerator (void)
  : Default Konstruktor
- CRobCodeGenerator (double speedIn, bool speedManualIn, bool orientationManualIn, tuple< double, double, double > angles)
  : Konstruktor
- ~CRobCodeGenerator (void)
  : Dekonstruktor
- void generateRobCode (vector< CInputPoint3D > &path, string filename)
  : Erstellt Roboter Code File
- void postProcessing (vector< CInputPoint3D > &path)
  : Nachbearbeitung der Daten
- double calculateSpeed (CInputPoint3D &p, size_t i, double timePrev)
  : Berechnet die Geschwindigkeit zwischen zwei Punkten
- void calculateAngles (COutputPoint3D &p, CInputPoint3D &pIn)
  : Berechnet die Geschwindigkeit zwischen zwei Punkten

Private Attribute

- vector< COutputPoint3D > processedPath
- double speed
- bool speedManual
- bool orientationManual
- double A
- double B
- double C

5.11.1 Ausführliche Beschreibung

: Klasse zum erstellen des Roboter Codes

In dieser Klasse wird die Nachbearbeitung der Daten mit den einstellbaren Daten durchgeführt.

Definiert in Zeile 23 der Datei RobCodeGenerator.h.

5.11.2 Beschreibung der Konstruktoren und Destruktoren

5.11.2.1 CRobCodeGenerator() [1/2]

```
CRobCodeGenerator::CRobCodeGenerator (
            void  )
```

: Default Konstruktor

Initialisiert der Daten

Siehe auch

: CRobCodeGenerator(double speedIn, bool speedManualIn, bool orientationManualIn, tuple<double, double, double> angles)

Definiert in Zeile 10 der Datei RobCodeGenerator.cpp.

```
00011 {
00012     speed = 0;
00013     speedManual = 0;
00014     orientationManual = 0;
00015     A = 0;
00016     B = 0;
00017     C = 0;
00018 }
```

5.11.2.2 CRobCodeGenerator() [2/2]

```
CRobCodeGenerator::CRobCodeGenerator (
            double speedIn,
            bool speedManualIn,
            bool orientationManualIn,
            tuple< double, double, double > angles )
```

: Konstruktor

Initialisiert der Daten

Parameter

	initSpeed double
	initSpeedManual bool
	initOrientationManual bool
	initA double
	initB double
	initC double

Siehe auch

: CRobCodeGenerator(void)

Definiert in Zeile 20 der Datei RobCodeGenerator.cpp.

```
00021 {
00022     speed = Speed;
00023     speedManual = SpeedManual;
00024     orientationManual = OrientationManual;
00025     A = get<0>(angles);
00026     B = get<1>(angles);
00027     C = get<2>(angles);
00028 }
```

5.11.2.3 ~CRobCodeGenerator()

```
CRobCodeGenerator::~CRobCodeGenerator (
            void  )
```

: Dekonstruktor

Definiert in Zeile 30 der Datei RobCodeGenerator.cpp.

```
00031 {
00032 }
```

5.11.3 Dokumentation der Elementfunktionen

5.11.3.1 calculateAngles()

```
void CRobCodeGenerator::calculateAngles (
            COutputPoint3D & p,
            CInputPoint3D & pIn )
```

: Berechnet die Geschwindigkeit zwischen zwei Punkten

Parameter

	COutputPoint3D& p
	CInputPoint3D& pIn

Definiert in Zeile 129 der Datei RobCodeGenerator.cpp.

```
00130 {
00131     // Funktion in Eulermatrix aufrufen die a/b/c neu berechnet
00132
00133     CEulerMatrix matrix = pIn.getEulerMatrix();
00134     tuple<double, double, double> abc;
00135
00136     abc = matrix.calculateAngels();
00137
00138     p.setA(get<0>(abc));
00139     p.setB(get<1>(abc));
00140     p.setC(get<2>(abc));
00141 }
```

5.11.3.2 calculateSpeed()

```
double CRobCodeGenerator::calculateSpeed (
            CInputPoint3D & p,
            size_t i,
            double timePrev )
```

: Berechnet die Geschwindigkeit zwischen zwei Punkten

Parameter

	CInputPoint3D& p aktueller Punkt
	size_t i Position im processedPath
	double timePrev Zeitstempel des vorherigen Punkts

Rückgabe

:

Definiert in Zeile 113 der Datei RobCodeGenerator.cpp.

```
00114 {
00115     double distance = 0;
00116     double time = 0;
00117
00118     distance = processedPath[s - 1].distanceTo(p); //Strecke zwischen p und dem Punkt zuvor
00119     time = p.getTime() - timePrev; //Zeit zwischen p-1 und p
00120
00121     speed = distance / time; // Berechnug Geschwindigkeit zwischen zwei Punkten
00122
00123     if (speed > MAX_SPEED) //Begrenzung auf maximale Geschwindigkeit, falls Trackerdaten hï¿½heren
      Wert aufweisen
00124         speed = MAX_SPEED;
00125
00126     return speed; //Zuweisung der Geschwindigkeit
00127 }
```

5.11.3.3 generateRobCode()

```
void CRobCodeGenerator::generateRobCode (
            vector< CInputPoint3D > & path,
            string filename )
```

: Erstellt Roboter Code File

Ruft das Postprocessing auf und speichert die bearbeiteten Daten als .src File ab

Parameter

	vector<CInputPoint3D>& path
	string filename

Definiert in Zeile 34 der Datei RobCodeGenerator.cpp.

```
00035 {
00036     postProcessing(points); // Calculates all the necessary values
00037
00038     errno_t err;
00039
00040     FILE* fid;
00041
00042     if ((err = fopen_s(&fid, filename.c_str(), "w")) != 0) // Errorhandling for File opening
00043     {
00044         string msg = "Open file: ";
00045         msg += filename;
00046         msg += " failed!";
00047
00048         throw exception(msg.c_str());
00049     }
00050
00051     COutputPoint3D currentPoint;
00052
00053     filename.erase(filename.end()-4,filename.end());
00054     fprintf(fid, "DEF %s \n", filename.c_str());
00055
00056     fputs("PTP $POS_ACT\n", fid);
00057
00058     if (speedManual) // If the speed is set to manual, it will be defined once at the beginning of the
      file
00059     {
00060         fprintf(fid, "&VEL.CP %f\n", speed);
00061     }
00062
00063     for (size_t s = 0; s < points.size(); s++)
00064     {
00065         currentPoint.set(points[s].getX(),points[s].getY(),points[s].getZ());
00066
00067         if (!speedManual) // If the speed is calculated it needs to be before every LIN command
00068             fprintf(fid, "&VEL.CP %f\n", currentPoint.getSpeed());
00069         fprintf(fid, "LIN {X %f, Y %f, Z %f, A %f, B %f, C %f}\n", currentPoint.getX(),
      currentPoint.getY(), currentPoint.getZ(),
00070             currentPoint.getA(), currentPoint.getB(), currentPoint.getC());
00071     }
00072
00073     fputs("END", fid);
00074 }
```

5.11.3.4 postProcessing()

```
void CRobCodeGenerator::postProcessing (
            vector< CInputPoint3D > & path )
```

: Nachbearbeitung der Daten

Integration der Manuellen Eingabedaten in die eingelesenen und bearbeiteten Daten Hier werden calculateSpeed und calculateAngles aufgerufen.

Parameter

	vector<CInputPoint3D>& path

Definiert in Zeile 76 der Datei RobCodeGenerator.cpp.

```
00077 {
00078     COutputPoint3D p;
00079     CInputPoint3D  pIn;
00080     double timePrev = 0;
00081 
00082     for (size_t s = 0; s < path.size(); s++) // FÃ¼r jeden Punkt in dem Vector
00083     {
00084         p.set(path[s].getX(), path[s].getY(), path[s].getZ());
00085         if (speedManual)
00086         {
00087             if (speed > MAX_SPEED) //Wenn maximale Geschwindigkeit ï¿½berschritten wird,
      Geschwindigkeit begrenzen
00088                 speed = MAX_SPEED;
00089         }
00090         else
00091         {
00092             if (s == 0)
00093                 p.setSpeed(1); //Der erste Punkt(0) wird mit Standardgeschwindigkeit 1m/s angefahren.
00094 
00095             else
00096                 p.setSpeed(calculateSpeed(path[s], s, timePrev)); //Die Geschwindigkeit zwischen den
      weiteren Punkten wird berechnet.
00097         }
00098 
00099         if (!orientationManual) // Wenn der Winkel vorgegeben ist diesen setzten
00100         {
00101             p.setA(A);
00102             p.setB(B);
00103             p.setC(C);
00104         }
00105         else // Sonst den Winkel berechnen
00106             calculateAngles(p, pIn);
00107         timePrev = path[s].getTime();
00108         processedPath.push_back(p);
00109     }
00110 
00111 }
```

5.11.4 Dokumentation der Datenelemente

5.11.4.1 A

```
double CRobCodeGenerator::A  [private]
```

User eingegebener Winkel A

Definiert in Zeile 98 der Datei RobCodeGenerator.h.

5.11.4.2 B

```
double CRobCodeGenerator::B  [private]
```

User eingegebener Winkel B

Definiert in Zeile 102 der Datei RobCodeGenerator.h.

5.11.4.3 C

```
double CRobCodeGenerator::C  [private]
```

User eingegebener Winkel C

Definiert in Zeile 106 der Datei RobCodeGenerator.h.

5.11.4.4 orientationManual

```
bool CRobCodeGenerator::orientationManual  [private]
```

Auswahl ob berechnete oder eingegebene Winkel verwendet werden soll

Definiert in Zeile 94 der Datei RobCodeGenerator.h.

5.11.4.5 processedPath

```
vector<COutputPoint3D> CRobCodeGenerator::processedPath  [private]
```

Fertig bearbeiteter Pfad

Definiert in Zeile 82 der Datei RobCodeGenerator.h.

5.11.4.6 speed

```
double CRobCodeGenerator::speed  [private]
```

User eingegebene Geschwindigkeit

Definiert in Zeile 86 der Datei RobCodeGenerator.h.

5.11.4.7 speedManual

```
bool CRobCodeGenerator::speedManual  [private]
```

Auswahl ob berechnete oder eingegebene Geschwindigkeit verwendet werden soll

Definiert in Zeile 90 der Datei RobCodeGenerator.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/RobCodeGenerator.h
- src/RobCodeGenerator.cpp

5.12 CSegmentApproximator Klassenreferenz

: Ausdünnen des Pfades

```
#include <SegmentApproximator.h>
```

Öffentliche Methoden

- CSegmentApproximator (void)
  : Default Konstruktor
- ~CSegmentApproximator (void)
  : Dekonstruktor
- void approx (const vector< list< CInputPoint3D > > &Segments, CLogging log)
  : Ruft die Douglas-Peuker Funktion auf
- void setmaxDistance (double maxDistanceSource)
  : Setzt die maximale Abweichung
- double getmaxDistance ()
  : Gibt die maximale Abweichung zurück
- vector< list< CInputPoint3D > > & getSegmentsApproxVector ()
  : Gibt den bereinigten Pfad zurück

Private Methoden

- void douglasPeuckerRecursive (list< CInputPoint3D > &segment, std::list< CInputPoint3D >::iterator startItr, std::list< CInputPoint3D >::iterator endItr)
  : Rekursive Douglas Peuker Funktion

Private Attribute

- vector< list< CInputPoint3D > > segmentsApprox
- double maxDistance

5.12.1 Ausführliche Beschreibung

: Ausdünnen des Pfades

In dieser Klasse wird der Pfad mit hilfe des Douglas-Peuker Algorithmusses ausgedünnt

Definiert in Zeile 24 der Datei SegmentApproximator.h.

5.12.2 Beschreibung der Konstruktoren und Destruktoren

5.12.2.1 CSegmentApproximator()

```
CSegmentApproximator::CSegmentApproximator (
            void  )
```

: Default Konstruktor

Initialisiert die Klasse

Definiert in Zeile 11 der Datei SegmentApproximator.cpp.

```
00012 {
00013 }
```

5.12.2.2 ~CSegmentApproximator()

```
CSegmentApproximator::~CSegmentApproximator (
            void  )
```

: Dekonstruktor

Definiert in Zeile 15 der Datei SegmentApproximator.cpp.

```
00016 {
00017 }
```

5.12.3 Dokumentation der Elementfunktionen

5.12.3.1 approx()

```
void CSegmentApproximator::approx (
            const vector< list< CInputPoint3D > > & Segments,
            CLogging log )
```

: Ruft die Douglas-Peuker Funktion auf

Iteriert durch die Listen im Vektor und ruft die Douglas-Peuker-Funktion auf

Parameter

	Segments const vector<list<CInputPoint3D>>&
	CLogging log für das Logging

Definiert in Zeile 19 der Datei SegmentApproximator.cpp.

```
00020 {
00021     CInputPoint3D p;
00022
00023     segmentsApprox = segments;
00024
00025     for (size_t s = 0; s < segments.size(); s++)
00026     {
00027         douglasPeuckerRecursive(segmentsApprox[s], segmentsApprox[s].begin(),
      --(segmentsApprox[s].end()));
00028     }
00029     log.setStep(3);
00030     log.logData(segmentsApprox);
00031 }
```

5.12.3.2 douglasPeuckerRecursive()

```
void CSegmentApproximator::douglasPeuckerRecursive (
            list< CInputPoint3D > & segment,
            std::list< CInputPoint3D >::iterator startItr,
            std::list< CInputPoint3D >::iterator endItr )  [private]
```

: Rekursive Douglas Peuker Funktion

Rekursive Funktion die durch das Segment geht und Punkte aus dem Pfad löscht wenn ihr Abstand zu groß wird.

Parameter

	list<CInputPoint3D>& segment

Parameter

	std::list<CInputPoint3D>::iterator startItr
	std::list<CInputPoint3D>::iterator endItr

Definiert in Zeile 49 der Datei SegmentApproximator.cpp.

```
00050 {
00051     if (segment.size() < 3) return;
00052     if (distance(startItr, endItr) == 2) return;
00053     CInputPoint3D pStart; CInputPoint3D pEnd;
00054     pStart.setX(startItr->getX()); pStart.setY(startItr->getY()); pStart.setZ(startItr->getZ());
00055
00056     pEnd.setX(endItr->getX()); pEnd.setY(endItr->getY()); pEnd.setZ(endItr->getZ());
00057
00058     double dist = 0.0, maxDist = 0.0;
00059     std::list<CInputPoint3D>::iterator maxItr, itr;
00060
00061     for (itr = startItr; itr != endItr; itr++)
00062     {
00063         CLine3D line = CLine3D(pStart, pEnd);
00064         // calc distance
00065         dist = itr->distanceTo(line);
00066         if (dist > maxDist) {
00067             maxDist = dist;
00068             maxItr = itr;
00069         }
00070     }
00071
00072     if (maxDist <= maxDistance) {
00073
00074         segment.erase((++startItr), endItr);
00075         return;
00076     }
00077
00078     douglasPeuckerRecursive(segment, startItr, maxItr);
00079     douglasPeuckerRecursive(segment, maxItr, endItr);
00080 }
```

5.12.3.3 getmaxDistance()

```
double CSegmentApproximator::getmaxDistance ( )
```

: Gibt die maximale Abweichung zurück

Rückgabe

: maxDistanceSource double

Definiert in Zeile 38 der Datei SegmentApproximator.cpp.

```
00039 {
00040     return maxDistance;
00041 }
```

5.12.3.4 getSegmentsApproxVector()

```
vector< list< CInputPoint3D > > & CSegmentApproximator::getSegmentsApproxVector ( )
```

: Gibt den bereinigten Pfad zurück

Rückgabe

: vector<list<CInputPoint3D>>&

Definiert in Zeile 43 der Datei SegmentApproximator.cpp.

```
00044 {
00045     return segmentsApprox;
00046 }
```

5.12.3.5 setmaxDistance()

```
void CSegmentApproximator::setmaxDistance (
            double maxDistanceSource )
```

: Setzt die maximale Abweichung

Parameter

	maxDistanceSource double

Definiert in Zeile 33 der Datei SegmentApproximator.cpp.

```
00034 {
00035     maxDistance = maxDistanceSource;
00036 }
```

5.12.4 Dokumentation der Datenelemente

5.12.4.1 maxDistance

```
double CSegmentApproximator::maxDistance  [private]
```

Einstellbare Distanz für den Douglas-Peuker-Algorithmus

Definiert in Zeile 69 der Datei SegmentApproximator.h.

5.12.4.2 segmentsApprox

```
vector<list<CInputPoint3D> > CSegmentApproximator::segmentsApprox  [private]
```

Bereinigten Pfad

Definiert in Zeile 65 der Datei SegmentApproximator.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- header/SegmentApproximator.h
- src/SegmentApproximator.cpp

Kapitel 6

Datei-Dokumentation

6.1 header/EulerMatrix.h-Dateireferenz

: Header File handling Euler Matrix

```
#include <tuple>
#include <cmath>
```

Klassen

- class CEulerMatrix

  : Handling und Berechnung Euler Matrix

6.1.1 Ausführliche Beschreibung

: Header File handling Euler Matrix

Definiert in Datei EulerMatrix.h.

6.2 EulerMatrix.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include <tuple>
00008 #include <cmath>
00009
00010 using namespace std;
00011
00012 #pragma once
00013
00018 class CEulerMatrix
00019 {
00020 public:
00026     CEulerMatrix(void);
00033     CEulerMatrix(float inputMatrix[3][3]);
00037     ~CEulerMatrix();
00038
00043     void setMatrix(float inputMatrix[3][3]);
00048     CEulerMatrix getEulerMatrix(void);
00049
00054     void getMatrix(float Matrix[][3]);
00062     CEulerMatrix angels2mat(double A, double B, double C);
00063
00068     tuple<double , double , double> calculateAngels(void);
00069
00070 private:
00074     float eulerMatrix[3][3];
00075 };
00076
```

6.3 header/GUI.h-Dateireferenz

Klassen

- class CGUI

6.4 GUI.h

gehe zur Dokumentation dieser Datei

```
00001 #pragma once
00002
00003 class CGUI
00004 {
00005
00006 public:
00007     CGUI();
00008     ~CGUI();
00009 };
```

6.5 header/InputParameter.h-Dateireferenz

: Header File Daten Einlesen

```
#include ¨EulerMatrix.h¨
#include ¨Point3D.h¨
#include <string>
#include <vector>
#include <list>
#include <iostream>
#include <fstream>
#include <sstream>
#include <tuple>
```

Klassen

- class CInputParameter

  : Handling Eingabedaten

6.5.1 Ausführliche Beschreibung

: Header File Daten Einlesen

Definiert in Datei InputParameter.h.

6.6 InputParameter.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "EulerMatrix.h"
00008 #include "Point3D.h"
00009 #include <string>
00010 #include <vector>
00011 #include <list>
00012 #include <iostream>
00013 #include <fstream>
00014 #include <sstream>
00015 #include <tuple>
00016
00017 using namespace std;
00018
00019 #pragma once
00020
00025 class CInputParameter
00026 {
00027 public:
00033     CInputParameter(void);
00047     CInputParameter(double initSpeed, bool initSeepManual, bool initOrientationManual, double initA,
      double initB, double initC);
00051     ~CInputParameter(void);
00052
00061     void setOrientation(bool initOrientationManual, double initA, double initB, double initC);
00068     void setSpeed(double initSpeed, bool initSpeedManual);
00069
00074     double getSpeed(void);
00079     bool getSpeedManual(void);
00084     bool getOrientationManual(void);
00090     tuple <double, double, double> getAngles(void);
00091
00097     void openFile(std::string path);
00106     bool detectJump(CInputPoint3D p, double x_prev, double y_prev, double z_prev);
00111     vector<list<CInputPoint3D>& getPath();
00112
00113 private:
00117     vector<list<CInputPoint3D> initialPath;
00121     double speed;
00125     bool speedManual;
00129     bool orientationManual;
00133     double A;
00137     double B;
00141     double C;
00145     double difference = 20;
00146 };
00147
```

6.7 header/Line3D.h-Dateireferenz

: Header File Daten Einlesen

```
#include ¨Point3D.h¨
#include <math.h>
```

Klassen

- class CLine3D

  : Berechnung Geraden

6.7.1 Ausführliche Beschreibung

: Header File Daten Einlesen

Definiert in Datei Line3D.h.

6.8 Line3D.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "Point3D.h"
00008 #include <math.h>
00009
00010 using namespace std;
00011
00012 #pragma once
00013
00018 class CLine3D
00019 {
00020 public:
00026     CLine3D(void);
00032     CLine3D(CPoint3D P1, CPoint3D P2);
00036     ~CLine3D(void);
00037
00041     CPoint3D p1;
00045     CPoint3D p2;
00046 };
00047
```

6.9 header/Logging.h-Dateireferenz

: Logging der Daten

```
#include ¨EulerMatrix.h¨
#include ¨Point3D.h¨
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <list>
```

Klassen

- class CLogging

  : Gleitender Mittelwertfilter

6.9.1 Ausführliche Beschreibung

: Logging der Daten

Definiert in Datei Logging.h.

6.10 Logging.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "EulerMatrix.h"
00008 #include "Point3D.h"
00009
00010 #include <iostream>
00011 #include <fstream>
00012 #include <sstream>
00013 #include <string>
00014 #include <vector>
00015 #include <list>
00016
00017 #pragma once
00022 class CLogging
00023 {
00024 public:
00030     CLogging(void);
00036     CLogging(string path);
00040     ~CLogging(void);
00045     void setStep(int Step);
00051     void logData(vector<list<CInputPoint3D»& sourcePath);
00057     void logData(vector<CInputPoint3D>& sourcePath);
00058 private:
00062     int step;
00066     string path;
00067 };
00068
00069
```

6.11 header/MeanFilter.h-Dateireferenz

: Berechnung des gleitenden Mittelwertfilters

```
#include <vector>
#include <list>
#include <string>
#include ¨Point3D.h¨
#include ¨Logging.h¨
```

Klassen

- class CMeanFilter

  : Gleitender Mittelwertfilter

6.11.1 Ausführliche Beschreibung

: Berechnung des gleitenden Mittelwertfilters

Definiert in Datei MeanFilter.h.

6.12 MeanFilter.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include <vector>
00008 #include <list>
00009 #include <string>
00010 #include "Point3D.h"
00011 #include "Logging.h"
00012
00013 #pragma once
00014
00015 using namespace std;
00016
00021 class CMeanFilter
00022 {
00023 public:
00029     CMeanFilter();
00036     CMeanFilter(int Window);
00040     ~CMeanFilter();
00041
00046     void setWindowSize(int Window);
00047
00052     int getWindowSize();
00057     vector<list<CInputPoint3D»& getPath();
00058
00065     list<CInputPoint3D> calculateMean(list<CInputPoint3D>& segment);
00073     void mean(vector<list<CInputPoint3D»& sourcePath, CLogging log);
00074
00075 private:
00079     int windowSize;
00083     vector<list<CInputPoint3D» meanPath;
00084 };
00085
```

6.13 header/PathBuilder.h-Dateireferenz

: Setzt die einzelnen Segmente zu einem Vector zusammen

```
#include <vector>
#include <list>
#include <iostream>
#include ¨Point3D.h¨
#include ¨Logging.h¨
```

Klassen

- class CPathBuilder

  : Zusammensetzten des Pfades

6.13.1 Ausführliche Beschreibung

: Setzt die einzelnen Segmente zu einem Vector zusammen

Definiert in Datei PathBuilder.h.

6.14 PathBuilder.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include <vector>
00008 #include <list>
00009 #include <iostream>
00010 #include "Point3D.h"
00011 #include "Logging.h"
00012
00013 using namespace std;
00014
00015 #pragma once
00016
00021 class CPathBuilder
00022 {
00023 public:
00028     CPathBuilder(void);
00032     ~CPathBuilder(void);
00033
00038     vector<CInputPoint3D>& getPath();
00044     void createPath(vector<list<CInputPoint3D»& segments, CLogging log);
00045
00046 private:
00050     vector<CInputPoint3D> path;
00051 };
00052
```

6.15 header/Point3D.h-Dateireferenz

: Verarbeitung der Punkte

```
#include ¨EulerMatrix.h¨
```

Klassen

- class CPoint3D
  : Grundklasse Punkt
- class CInputPoint3D
  : Input Punkt
- class COutputPoint3D
  : Output Punkt

6.15.1 Ausführliche Beschreibung

: Verarbeitung der Punkte

Definiert in Datei Point3D.h.

6.16 Point3D.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "EulerMatrix.h"
00008
00009 class CLine3D;
00010
00011 using namespace std;
00012
00013 #pragma once
00014
00020 class CPoint3D
00021 {
00022 public:
00028     CPoint3D(void);
00037     CPoint3D(double X, double Y, double Z);
00041     ~CPoint3D(void);
00042
00047     double getX();
00052     double getY();
00057     double getZ();
00058
00063     void setX(double X);
00068     void setY(double Y);
00073     void setZ(double Z);
00074
00081     void set(double X, double Y, double Z);
00087     double distanceTo(CPoint3D point);
00093     double distanceTo(CLine3D line);
00094
00095 protected:
00099     double x, y, z;
00100 };
00101
00106 class CInputPoint3D : public CPoint3D
00107 {
00108 public:
00114     CInputPoint3D(void);
00125     CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix);
00129     ~CInputPoint3D(void);
00130
00135     double getTime();
00140     CEulerMatrix getEulerMatrix();
00141
00146     void setTime(double time);
00151     void setEulerMatrix(CEulerMatrix orientation);
00160     void setPoint(double time, double X, double Y, double Z, CEulerMatrix orientation);
00161
00162 private:
00166     double timestamp;
00170     CEulerMatrix orientationMatrix;
00171 };
00172
00177 class COutputPoint3D : public CPoint3D
00178 {
00179 public:
00185     COutputPoint3D(void);
00198     COutputPoint3D(double Speed, double X, double Y, double Z, double A, double B, double C);
00202     ~COutputPoint3D(void);
00203
00208     double getSpeed();
00213     double getA();
00218     double getB();
00223     double getC();
00224
00229     void setSpeed(double speed);
00234     void setA(double A);
00239     void setB(double B);
00244     void setC(double C);
00245 private:
00249     double a, b, c;
00253     double speed;
00254 };
```

6.17 header/RobCodeGenerator.h-Dateireferenz

: Erstellung des Roboter Codes

```
#include <vector>
#include <iostream>
#include ¨Point3D.h¨
#include <tuple>
```

Klassen

- class CRobCodeGenerator

  : Klasse zum erstellen des Roboter Codes

Makrodefinitionen

- #define MAX_SPEED 2.0

6.17.1 Ausführliche Beschreibung

: Erstellung des Roboter Codes

Definiert in Datei RobCodeGenerator.h.

6.17.2 Makro-Dokumentation

6.17.2.1 MAX_SPEED

```
#define MAX_SPEED 2.0
```

Definiert in Zeile 16 der Datei RobCodeGenerator.h.

6.18 RobCodeGenerator.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include <vector>
00008 #include <iostream>
00009 #include "Point3D.h"
00010 #include <tuple>
00011
00012 using namespace std;
00013
00014 #pragma once
00015
00016 #define MAX_SPEED 2.0
00017
00023 class CRobCodeGenerator
00024 {
00025 public:
00031     CRobCodeGenerator(void);
00043     CRobCodeGenerator(double speedIn, bool speedManualIn, bool orientationManualIn, tuple<double,
      double, double> angles);
00047     ~CRobCodeGenerator(void);
00048
00055     void generateRobCode(vector<CInputPoint3D>& path, string filename);
00062     void postProcessing(vector<CInputPoint3D>& path);
00070     double calculateSpeed(CInputPoint3D& p, size_t i, double timePrev);
00076     void calculateAngles(COutputPoint3D& p, CInputPoint3D& pIn);
00077
00078 private:
00082     vector<COutputPoint3D> processedPath;
00086     double speed;
00090     bool speedManual;
00094     bool orientationManual;
00098     double A;
00102     double B;
00106     double C;
00107
00108 };
00109
```

6.19 header/SegmentApproximator.h-Dateireferenz

: Berechnung des Douglas Peuker Algorithmusses

```
#include <vector>
#include <list>
#include <iostream>
#include <math.h>
#include ¨Point3D.h¨
#include ¨Logging.h¨
```

Klassen

- class CSegmentApproximator

  : Ausdünnen des Pfades

6.19.1 Ausführliche Beschreibung

: Berechnung des Douglas Peuker Algorithmusses

Definiert in Datei SegmentApproximator.h.

6.20 SegmentApproximator.h

gehe zur Dokumentation dieser Datei

```
00001
00007 //TODO: Kommentare
00008
00009 #include <vector>
00010 #include <list>
00011 #include <iostream>
00012 #include <math.h>
00013 #include "Point3D.h"
00014 #include "Logging.h"
00015
00016 using namespace std;
00017
00018 #pragma once
00019
00024 class CSegmentApproximator
00025 {
00026 public:
00031     CSegmentApproximator(void);
00035     ~CSegmentApproximator(void);
00036
00043     void approx(const vector<list<CInputPoint3D»& Segments, CLogging log);
00048     void setmaxDistance(double maxDistanceSource);
00053     double getmaxDistance();
00054
00059     vector<list<CInputPoint3D»& getSegmentsApproxVector();
00060
00061 private:
00065     vector<list<CInputPoint3D» segmentsApprox;
00069     double maxDistance;
00070
00079     void douglasPeuckerRecursive(list<CInputPoint3D>& segment, std::list<CInputPoint3D>::iterator
      startItr, std::list<CInputPoint3D>::iterator endItr);
00080 };
```

6.21 src/EulerMatrix.cpp-Dateireferenz

: Source Code der Euler Matrix

```
#include ¨./header/EulerMatrix.h¨
#include <math.h>
```

6.21.1 Ausführliche Beschreibung

: Source Code der Euler Matrix

Definiert in Datei EulerMatrix.cpp.

6.22 EulerMatrix.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/EulerMatrix.h"
00008 #include <math.h>
00009
00010 CEulerMatrix::CEulerMatrix(void)
00011 {
00012     for (int i = 0; i < 3; i++)
00013     {
00014         for (int m = 0; m < 3; m++)
00015         {
00016             eulerMatrix[i][m] = 0;                  // eulerMatrix mit 0 initialisieren
00017         }
00018     }
00019 }
00020
00021 CEulerMatrix::CEulerMatrix(float inputMatrix[3][3])
00022 {
00023     for (int i = 0; i < 3; i++)
00024     {
00025         for (int m = 0; m < 3; m++)
00026         {
00027             eulerMatrix[i][m] = inputMatrix[i][m];  // eulerMatrix mit Startwerten initialisieren
00028         }
00029     }
00030 }
00031
00032 CEulerMatrix::~CEulerMatrix()
00033 {
00034 }
00035
00036
00037 void CEulerMatrix::setMatrix(float inputMatrix[3][3])
00038 {
00039     for(int i = 0; i < 3; i++)
00040     {
00041         for (int m = 0; m < 3; m++)
00042         {
00043             eulerMatrix[i][m] = inputMatrix[i][m];  // eulerMatrix mit Ãbergabewerten Ã¼berschreiben
00044         }
00045     }
00046 }
00047
00048 CEulerMatrix CEulerMatrix::getEulerMatrix()
00049 {
00050     return eulerMatrix;
00051 }
00052
00053 void CEulerMatrix::getMatrix(float Matrix[][3])
00054 {
00055     for (int i = 0; i < 3; i++)
00056     {
00057         for (int m = 0; m < 3; m++)
00058         {
00059             Matrix[i][m] = eulerMatrix[i][m];  // eulerMatrix mit ï¿½bergabewerten ï¿½berschreiben
```

```
00060         }
00061     }
00062 }
00063
00064 //TODO: Kommentar
00065 CEulerMatrix CEulerMatrix::angels2mat(double A, double B, double C)
00066 {
00067     float Matrix[3][3];
00068
00069     Matrix[0][0] = cos(A) * cos(C) - sin(A) * cos(B) * sin(C);
00070     Matrix[0][1] = -cos(A) * sin(C) - sin(A) * cos(B) * cos(C);
00071     Matrix[0][2] = sin(A) * sin(B);
00072
00073     Matrix[1][0] = sin(A) * cos(C) + cos(A) * cos(B) * sin(C);
00074     Matrix[1][1] = -sin(A) * sin(C) + cos(A) * cos(B) * cos(C);
00075     Matrix[1][2] = -cos(A) * sin(B);
00076
00077     Matrix[2][0] = sin(B) * sin(C);
00078     Matrix[2][1] = sin(B) * cos(C);
00079     Matrix[2][2] = cos(B);
00080
00081     CEulerMatrix buffer(Matrix);
00082     return buffer;
00083 }
00084
00085 //TODO: Kommentar
00086 tuple<double , double , double> CEulerMatrix::calculateAngels(void)
00087 {
00088     double a, b, c, sin_a, cos_a, sin_b, abs_cos_b, sin_c, cos_c;
00089
00090     a = atan2(eulerMatrix[1][0], eulerMatrix[0][0]);
00091
00092     sin_a = sin(a);
00093     cos_a = cos(a);
00094     sin_b = eulerMatrix[2][0] * -1;
00095     abs_cos_b = cos(a) * eulerMatrix[0][0] + sin(a) * eulerMatrix[1][0];
00096
00097     b = atan2 (sin_b, abs_cos_b);
00098
00099     sin_c =  sin_a * eulerMatrix[0][2] - cos_a * eulerMatrix[1][2];
00100     cos_c = -sin_a * eulerMatrix[0][1] + cos_a * eulerMatrix[1][1];
00101
00102    c = atan2(sin_c, cos_c);
00103
00104    return make_tuple(a, b, c);
00105 }
00106
```

6.23 src/GUI.cpp-Dateireferenz

```
#include ¨./header/GUI.h¨
```

6.24 GUI.cpp

gehe zur Dokumentation dieser Datei

```
00001 #include "./header/GUI.h"
00002
00003 CGUI::CGUI()
00004 {}
00005
00006 CGUI::~CGUI()
00007 {}
```

6.25 src/InputParameter.cpp-Dateireferenz

: Source File Daten Einlesen

```
#include ¨./header/InputParameter.h¨
#include ¨./header/Point3D.h¨
#include ¨./header/EulerMatrix.h¨
```

6.25.1 Ausführliche Beschreibung

: Source File Daten Einlesen

Definiert in Datei InputParameter.cpp.

6.26 InputParameter.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/InputParameter.h"
00008 #include "./header/Point3D.h"
00009 #include "./header/EulerMatrix.h"
00010
00014 CInputParameter::CInputParameter(double initSpeed, bool initSpeedManual, bool initOrientationManual,
      double initA, double initB, double initC)
00015 {
00016     speed = initSpeed;
00017     speedManual = initSpeedManual;
00018     orientationManual = initOrientationManual;
00019     A = initA;
00020     B = initB;
00021     C = initC;
00022
00023 }
00027 CInputParameter::CInputParameter(void)
00028 {
00029     speed = 0;
00030     A = 0;
00031     B = 0;
00032     C = 0;
00033     speedManual = false,
00034     orientationManual = false;
00035
00036 }
00040 CInputParameter::~CInputParameter(void)
00041 {
00042
00043 }
00044
00045 void CInputParameter::setOrientation(bool initOrientationManual, double initA, double initB, double
      initC)
00046 {
00047     orientationManual = initOrientationManual;
00048     A = initA;
00049     B = initB;
00050     C = initC;
00051 }
00052
00053 void CInputParameter::setSpeed(double initSpeed, bool initSpeedManual)
00054 {
00055     speed = initSpeed;
00056     speedManual = initSpeedManual;
00057 }
00058
00059 vector<list<CInputPoint3D>& CInputParameter::getPath()
00060 {
00061     return initialPath;
00062 }
00063
00064 double CInputParameter::getSpeed(void)
00065 {
00066     return speed;
00067 }
00068
00069 bool CInputParameter::getSpeedManual(void)
00070 {
00071     return speedManual;
00072 }
00073
00074 bool CInputParameter::getOrientationManual(void)
00075 {
00076     return orientationManual;
00077 }
00078
00079 tuple <double, double, double> CInputParameter::getAngles(void)
00080 {
00081     return make_tuple(A, B, C);
```

```
00082 }
00083
00084 //TODO: Kommentar
00085 void CInputParameter::openFile(string path)
00086 {
00087     ifstream fin(path);
00088     char delimiter = ' ';
00089     CInputPoint3D tmpPoint;
00090     CEulerMatrix tmpEuler;
00091     double x, y, z;
00092     double x_prev = 0, y_prev = 0, z_prev = 0;
00093     double timestamp;
00094     int segmentCount = -1;
00095     float dummyMatrix[3][3];
00096
00097
00098     if (!fin.is_open())
00099     {
00100         cerr « "Datei konnte nicht geöffnet werden" « endl;
00101     }
00102     string line;
00103
00104     while(getline(fin, line))
00105     {
00106         std::istringstream sStream (line);
00107         sStream » timestamp » x » y » z » dummyMatrix[0][0] » dummyMatrix[0][1] » dummyMatrix[0][2]
00108             » dummyMatrix[1][0] » dummyMatrix[1][1] » dummyMatrix[1][2] » dummyMatrix[2][0] »
    dummyMatrix[2][1] » dummyMatrix[2][2];
00109
00110         tmpEuler.setMatrix(dummyMatrix);
00111         tmpPoint.setPoint(timestamp, x, y, z, tmpEuler.getEulerMatrix());
00112
00113         if (detectJump(tmpPoint, x_prev, y_prev, z_prev))
00114         {
00115             segmentCount++;
00116             initialPath.push_back(list<CInputPoint3D>());
00117         }
00118
00119         initialPath[segmentCount].push_back(tmpPoint);
00120
00121         x_prev = x;
00122         y_prev = y;
00123         z_prev = z;
00124     }
00125     fin.close();
00126 }
00127
00128 //TODO: Kommentar
00129 bool CInputParameter::detectJump(CInputPoint3D p, double x_prev, double  y_prev, double z_prev)
00130 {
00131     if(abs(p.getX() - x_prev) > difference)
00132         return true;
00133     else if(abs(p.getY() - y_prev) > difference)
00134         return true;
00135     else if(abs(p.getZ() - z_prev) > difference)
00136         return true;
00137     else
00138         return false;
00139 }
```

6.27 src/Line3D.cpp-Dateireferenz

: Source File Line3D

```
#include ¨./header/Line3D.h¨
#include ¨./header/Point3D.h¨
```

6.27.1 Ausführliche Beschreibung

: Source File Line3D

Definiert in Datei Line3D.cpp.

6.28 Line3D.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/Line3D.h"
00008 #include "./header/Point3D.h"
00009
00010 CLine3D::CLine3D(void)
00011 {
00012 }
00013
00014 CLine3D::CLine3D(CPoint3D P1, CPoint3D P2)
00015 {
00016     p1 = P1;
00017     p2 = P2;
00018 }
00019
00020 CLine3D::~CLine3D(void)
00021 {
00022 }
```

6.29 src/Logging.cpp-Dateireferenz

: Source File Logging

`#include ¨header/Logging.h¨`

6.29.1 Ausführliche Beschreibung

: Source File Logging

Definiert in Datei Logging.cpp.

6.30 Logging.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "header/Logging.h"
00008
00009 CLogging::CLogging(void)
00010 {
00011     step = 0;
00012 }
00013
00014 CLogging::CLogging(string Path)
00015 {
00016     path = Path;
00017 }
00018
00019 CLogging::~CLogging(void)
00020 {
00021
00022 }
00023
00024 void CLogging::setStep(int Step)
00025 {
00026     step = Step;
00027 }
00028
00029 //TODO: Kommentar
00030 void CLogging::logData(vector<list<CInputPoint3D»& sourcePath)
00031 {
00032     string filepath;
00033     float dummyMatrix[3][3];
00034     CEulerMatrix tmpEuler;
```

```
00035
00036     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00037
00038     FILE* fid = fopen(filepath.c_str(), "w");
00039
00040     if (fid == NULL)
00041     {
00042         cerr « "ERROR - Can NOT write to output file!\n";
00043         return;
00044     }
00045
00046     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00047         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00048         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00049
00050     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00051     {
00052         list<CInputPoint3D>::iterator itr = sourcePath[s].begin();
00053
00054         tmpEuler = itr->getEulerMatrix();
00055         tmpEuler.getMatrix(dummyMatrix);
00056
00057         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00058             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00059             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00060             dummyMatrix[2][2]);
00061
00062
00063         for (; itr != sourcePath[s].end(); itr++) //for all points in the segment
00064         {
00065             fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00066                 dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00067                 dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00068                 dummyMatrix[2][2]);
00069         }
00070
00071         itr--;
00072         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)itr->getTime(),
    (double)itr->getX(), (double)itr->getY(), (double)itr->getZ(),
00073             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00074             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00075             dummyMatrix[2][2]);
00076     }
00077
00078     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00079         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00080         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00081 }
00082
00083 //TODO: Kommentar
00084 void CLogging::logData(vector<CInputPoint3D>& sourcePath)
00085 {
00086     string filepath;
00087     float dummyMatrix[3][3];
00088     CEulerMatrix tmpEuler;
00089
00090     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00091
00092     FILE* fid = fopen(filepath.c_str(), "w");
00093
00094     if (fid == NULL)
00095     {
00096         cerr « "ERROR - Can NOT write to output file!\n";
00097         return;
00098     }
00099
00100     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00101         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00102         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00103
00104     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00105     {
00106         tmpEuler.getMatrix(dummyMatrix);
00107
00108         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n", (double)sourcePath[s].getTime(),
00109             (double)sourcePath[s].getX(), (double)sourcePath[s].getY(), (double)sourcePath[s].getZ(),
00110             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2], dummyMatrix[1][0],
00111             dummyMatrix[1][1], dummyMatrix[1][2], dummyMatrix[2][0], dummyMatrix[2][1],
00112             dummyMatrix[2][2]);
00113     }
00114
00115     fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f %f\n",
00116         (double)0, (double)0, (double)0, (double)0, (float)0, (float)0, (float)0,
00117         (float)0, (float)0, (float)0, (float)0, (float)0, (float)0);
00118 }
```

6.31 src/MeanFilter.cpp-Dateireferenz

: Source File gleitender Mittelwertfilter

```
#include ¨./header/MeanFilter.h¨
#include ¨./header/Logging.h¨
#include <math.h>
```

6.31.1 Ausführliche Beschreibung

: Source File gleitender Mittelwertfilter

Definiert in Datei MeanFilter.cpp.

6.32 MeanFilter.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/MeanFilter.h"
00008 #include "./header/Logging.h"
00009 #include <math.h>
00010
00011 CMeanFilter::CMeanFilter()
00012 {
00013     windowSize = 3;
00014 }
00015
00016 CMeanFilter::CMeanFilter(int Window)
00017 {
00018     windowSize = Window;
00019 }
00020
00021 CMeanFilter::~CMeanFilter()
00022 {
00023 }
00024
00025 void CMeanFilter::setWindowSize(int Window)
00026 {
00027     windowSize = Window;
00028 }
00029
00030 int CMeanFilter::getWindowSize()
00031 {
00032     return windowSize;
00033 }
00034
00035 vector<list<CInputPoint3D»& CMeanFilter::getPath()
00036 {
00037     return meanPath;
00038 }
00039
00040 void CMeanFilter::mean(vector<list<CInputPoint3D»& sourcePath, CLogging log)
00041 {
00042     list<CInputPoint3D> dummyList;
00043     for (size_t s = 0; s < sourcePath.size(); s++)
00044     {
00045         dummyList = calculateMean(sourcePath[s]);
00046         meanPath.push_back(dummyList);
00047     }
00048     log.setStep(2);
00049     log.logData(meanPath);
00050 }
00051
00052 list<CInputPoint3D> CMeanFilter::calculateMean(list<CInputPoint3D>& segment)
00053 {
00054     double sumX = 0, sumY = 0, sumZ = 0;
00055     double div = 0;
00056     int m = 0;
00057     int OffsetPos = 0;
00058     int OffsetNeg = 0;
```

```
00059
00060     CInputPoint3D p;
00061
00062     size_t inputSize = segment.size();
00063
00064     list<CInputPoint3D>::iterator it = segment.begin();
00065     list<CInputPoint3D> newSegment;
00066
00067     for (size_t i = 0; i < inputSize - windowSize; ++i)
00068     {
00069         sumX = 0, sumY = 0, sumZ = 0;
00070         div = 0;
00071         p.setTime(it->getTime());
00072         p.setEulerMatrix(it->getEulerMatrix());
00073         for (size_t j = i; j < i + windowSize; ++j)
00074         {
00075
00076             sumX += it->getX();
00077             sumY += it->getY();
00078             sumZ += it->getZ();
00079             div++;
00080             it++;
00081         }
00082         for (size_t index = windowSize; index > 0; index--)
00083         {
00084             it--;
00085         }
00086         p.set(sumX / div, sumY / div, sumZ / div);
00087         if(it != segment.end())
00088             it++;
00089         newSegment.push_back(p);
00090     }
00091     return newSegment;
00092 }
```

6.33 src/PathBuilder.cpp-Dateireferenz

: Source File Segmente zu Pfad

`#include ¨./header/PathBuilder.h¨`

6.33.1 Ausführliche Beschreibung

: Source File Segmente zu Pfad

Definiert in Datei PathBuilder.cpp.

6.34 PathBuilder.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/PathBuilder.h"
00008
00009 CPathBuilder::CPathBuilder(void)
00010 {
00011 }
00012
00013
00014 CPathBuilder::~CPathBuilder(void)
00015 {
00016 }
00017
00018 vector<CInputPoint3D>& CPathBuilder::getPath()
00019 {
00020     return path;
00021 }
00022
```

```
00023 void CPathBuilder::createPath(vector<list<CInputPoint3D»& segments, CLogging log)
00024 {
00025     CInputPoint3D point; //startpoint
00026     path.push_back(point);
00027
00028     for (size_t s = 0; s < segments.size(); s++) //for all segments
00029     {
00030         list<CInputPoint3D>::iterator itr = segments[s].begin();
00031
00032         point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ()); //point over start
      of segment
00033         path.push_back(point);
00034
00035         for (; itr != segments[s].end(); itr++) //for all points in the segment
00036         {
00037             point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ());
00038             path.push_back(point);
00039         }
00040
00041         itr--;
00042
00043         point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ()); //point over end of
      segment
00044         path.push_back(point);
00045     }
00046
00047     point.set(0, 0, 0); //endpoint (== startpoint)
00048     path.push_back(point);
00049
00050     log.setStep(4);
00051     log.logData(path);
00052 }
```

6.35 src/Point3D.cpp-Dateireferenz

: Source File Punkte

```
#include ¨./header/Point3D.h¨
#include ¨./header/Line3D.h¨
#include <math.h>
```

6.35.1 Ausführliche Beschreibung

: Source File Punkte

Definiert in Datei Point3D.cpp.

6.36 Point3D.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/Point3D.h"
00008 #include "./header/Line3D.h"
00009 #include <math.h>
00010
00011
00012 /* initialisieren des Punktes */
00013 CPoint3D::CPoint3D(void)
00014 {
00015     x = 0;
00016     y = 0;
00017     z = 0;
00018 }
00019
00020 CPoint3D::CPoint3D(double X, double Y, double Z)
00021 {
```

```
00022     x = X;
00023     y = Y;
00024     z = Z;
00025 }
00026
00027 CPoint3D::~CPoint3D(void)
00028 {
00029 }
00030
00031 double CPoint3D::getX(void)
00032 {
00033     return x;
00034 }
00035
00036 double CPoint3D::getY(void)
00037 {
00038     return y;
00039 }
00040
00041 double CPoint3D::getZ(void)
00042 {
00043     return z;
00044 }
00045
00046 void CPoint3D::setX(double X)
00047 {
00048     x = X;
00049 }
00050
00051 void CPoint3D::setY(double Y)
00052 {
00053     y = Y;
00054 }
00055
00056 void CPoint3D::setZ(double Z)
00057 {
00058     z = Z;
00059 }
00060
00061 void CPoint3D::set(double X, double Y, double Z)
00062 {
00063     x = X;
00064     y = Y;
00065     z = Z;
00066 }
00067
00068 double CPoint3D::distanceTo(CPoint3D point)
00069 {
00070     return sqrt(pow((double)(x - (double)point.getX()), 2) + pow((double)(y - (double)point.getY()),
      2) + pow((double)(z - (double)point.getZ()), 2));   // Pythagoras 3D
00071 }
00072
00073 double CPoint3D::distanceTo(CLine3D line)
00074 {
00075     double bx, by, bz, rv_sq, dist, vp1, vp2, vp3;          // Variablen Anlegen
00076
00077     /*
00078     Vermessen wird der Punkt selbst
00079
00080     bx, by, bz      == Vektordifferenz
00081     rv_sq           == Betrag des Linienvektors
00082     dist            == Distanz von Punkt zu Linie
00083     vp1, vp2, vp3   == Vektorprodukte
00084     */
00085
00086     int rvx = line.p1.x - line.p2.x;      // Parameter X des Linienvektor berechnen
00087     int rvy = line.p1.y - line.p2.y;      // Parameter Y des Linienvektor berechnen
00088     int rvz = line.p1.z - line.p2.z;      // Parameter Z des Linienvektor berechnen
00089
00090     rv_sq = sqrt(((double)rvx * (double)rvx) + ((double)rvy * (double)rvy) + ((double)rvz *
      (double)rvz));      // Betrag des Linienvektor berechnen
00091
00092     bx = x - (double)line.p1.x;             // X(Punkt) - X(Aufpunkt)
00093     by = y - (double)line.p1.y;             // Y(Punkt) - Y(Aufpunkt)
00094     bz = z - (double)line.p1.z;             // Z(Punkt) - Z(Aufpunkt)
00095
00096     vp1 = by * rvz - bz * rvy;              // Parameter X Vektorprodukt
00097     vp2 = bz * rvx - bx * rvz;              // Parameter Y Vektorprodukt
00098     vp3 = bx * rvy - by * rvx;              // Parameter Z Vektorprodukt
00099
00100     dist = sqrt(vp1 * vp1 + vp2 * vp2 + vp3 * vp3) / rv_sq; // Betrag des Vektors berechnen
00101
00102     return dist;
00103 }
00104
00105 // InputPoint3D
00106
```

```
00107 CInputPoint3D::CInputPoint3D(void) : CPoint3D()
00108 {
00109     timestamp = 0;
00110 }
00111
00112 CInputPoint3D::CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)
00113 {
00114     x = X;
00115     y = Y;
00116     z = Z;
00117     timestamp = Timestamp;
00118     orientationMatrix = Matrix;
00119
00120 }
00121
00122 CInputPoint3D::~CInputPoint3D(void)
00123 {
00124 }
00125
00126 void CInputPoint3D::setEulerMatrix(CEulerMatrix orientation)
00127 {
00128     orientationMatrix = orientation;
00129 }
00130
00131
00132 void CInputPoint3D::setPoint(double time, double X, double Y, double Z, CEulerMatrix orientation)
00133 {
00134     setTime(time);
00135     set(X, Y, Z);
00136     setEulerMatrix(orientation);
00137 }
00138
00139 void CInputPoint3D::setTime(double time)
00140 {
00141     timestamp = time;
00142 }
00143
00144 CEulerMatrix CInputPoint3D::getEulerMatrix()
00145 {
00146     return orientationMatrix;
00147 }
00148
00149 double CInputPoint3D::getTime()
00150 {
00151     return timestamp;
00152 }
00153
00154 // OutputPoint3D
00155
00156 COutputPoint3D::COutputPoint3D(void) : CPoint3D()
00157 {
00158     speed = 0;
00159     a = 0;
00160     b = 0;
00161     c = 0;
00162 }
00163
00164 COutputPoint3D::COutputPoint3D(double Speed, double X, double Y, double Z, double A, double B, double
    C)
00165 {
00166     speed = Speed;
00167     a = A;
00168     b = B;
00169     c = C;
00170     x = X;
00171     y = Y;
00172     z = Z;
00173 }
00174
00175 COutputPoint3D::~COutputPoint3D(void)
00176 {
00177
00178 }
00179
00180 double COutputPoint3D::getA(void)
00181 {
00182     return a;
00183 }
00184
00185 double COutputPoint3D::getB(void)
00186 {
00187     return b;
00188 }
00189
00190 double COutputPoint3D::getC(void)
00191 {
00192     return c;
```

```
00193 }
00194
00195 double COutputPoint3D::getSpeed(void)
00196 {
00197     return speed;
00198 }
00199
00200 void COutputPoint3D::setA(double A)
00201 {
00202     a = A;
00203 }
00204
00205 void COutputPoint3D::setB(double B)
00206 {
00207     b = B;
00208 }
00209
00210 void COutputPoint3D::setC(double C)
00211 {
00212     c = C;
00213 }
00214
00215 void COutputPoint3D::setSpeed(double Speed)
00216 {
00217     speed = Speed;
00218 }
```

6.37 src/RobCodeGenerator.cpp-Dateireferenz

: Source File Roboter Code Erstellung

```
#include ¨./header/RobCodeGenerator.h¨
#include ¨./header/Point3D.h¨
```

6.37.1 Ausführliche Beschreibung

: Source File Roboter Code Erstellung

Definiert in Datei RobCodeGenerator.cpp.

6.38 RobCodeGenerator.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/RobCodeGenerator.h"
00008 #include "./header/Point3D.h"
00009
00010 CRobCodeGenerator::CRobCodeGenerator(void)
00011 {
00012     speed = 0;
00013     speedManual = 0;
00014     orientationManual = 0;
00015     A = 0;
00016     B = 0;
00017     C = 0;
00018 }
00019
00020 CRobCodeGenerator::CRobCodeGenerator(double Speed, bool SpeedManual, bool OrientationManual,
      tuple<double, double, double> angles)
00021 {
00022     speed = Speed;
00023     speedManual = SpeedManual;
00024     orientationManual = OrientationManual;
00025     A = get<0>(angles);
00026     B = get<1>(angles);
00027     C = get<2>(angles);
```

```
00028 }
00029
00030 CRobCodeGenerator::~CRobCodeGenerator(void)
00031 {
00032 }
00033
00034 void CRobCodeGenerator::generateRobCode(vector<CInputPoint3D>& points, string filename)
00035 {
00036     postProcessing(points); // Calculates all the necessary values
00037
00038     errno_t err;
00039
00040     FILE* fid;
00041
00042     if ((err = fopen_s(&fid, filename.c_str(), "w")) != 0) // Errorhandling for File opening
00043     {
00044         string msg = "Open file: ";
00045         msg += filename;
00046         msg += " failed!";
00047
00048         throw exception(msg.c_str());
00049     }
00050
00051     COutputPoint3D currentPoint;
00052
00053     filename.erase(filename.end()-4,filename.end());
00054     fprintf(fid, "DEF %s \n", filename.c_str());
00055
00056     fputs("PTP $POS_ACT\n", fid);
00057
00058     if (speedManual) // If the speed is set to manual, it will be defined once at the beginning of the
      file
00059     {
00060         fprintf(fid, "&VEL.CP %f\n", speed);
00061     }
00062
00063     for (size_t s = 0; s < points.size(); s++)
00064     {
00065         currentPoint.set(points[s].getX(),points[s].getY(),points[s].getZ());
00066
00067         if (!speedManual) // If the speed is calculated it needs to be before every LIN command
00068             fprintf(fid, "&VEL.CP %f\n", currentPoint.getSpeed());
00069         fprintf(fid, "LIN {X %f, Y %f, Z %f, A %f, B %f, C %f}\n", currentPoint.getX(),
      currentPoint.getY(), currentPoint.getZ(),
00070             currentPoint.getA(), currentPoint.getB(), currentPoint.getC());
00071     }
00072
00073     fputs("END", fid);
00074 }
00075
00076 void CRobCodeGenerator::postProcessing(vector<CInputPoint3D>& path)
00077 {
00078     COutputPoint3D p;
00079     CInputPoint3D  pIn;
00080     double timePrev = 0;
00081
00082     for (size_t s = 0; s < path.size(); s++) // FÃ¼r jeden Punkt in dem Vector
00083     {
00084         p.set(path[s].getX(), path[s].getY(), path[s].getZ());
00085         if (speedManual)
00086         {
00087             if (speed > MAX_SPEED) //Wenn maximale Geschwindigkeit ï¿½berschritten wird,
      Geschwindigkeit begrenzen
00088                 speed = MAX_SPEED;
00089         }
00090         else
00091         {
00092             if (s == 0)
00093                 p.setSpeed(1); //Der erste Punkt(0) wird mit Standardgeschwindigkeit 1m/s angefahren.
00094
00095             else
00096                 p.setSpeed(calculateSpeed(path[s], s, timePrev)); //Die Geschwindigkeit zwischen den
      weiteren Punkten wird berechnet.
00097         }
00098
00099         if (!orientationManual) // Wenn der Winkel vorgegeben ist diesen setzten
00100         {
00101             p.setA(A);
00102             p.setB(B);
00103             p.setC(C);
00104         }
00105         else // Sonst den Winkel berechnen
00106             calculateAngles(p, pIn);
00107         timePrev = path[s].getTime();
00108         processedPath.push_back(p);
00109     }
00110
```

```
00111 }
00112
00113 double CRobCodeGenerator::calculateSpeed(CInputPoint3D& p, size_t s, double timePrev)
00114 {
00115     double distance = 0;
00116     double time = 0;
00117
00118     distance = processedPath[s - 1].distanceTo(p); //Strecke zwischen p und dem Punkt zuvor
00119     time = p.getTime() - timePrev; //Zeit zwischen p-1 und p
00120
00121     speed = distance / time; // Berechnug Geschwindigkeit zwischen zwei Punkten
00122
00123     if (speed > MAX_SPEED) //Begrenzung auf maximale Geschwindigkeit, falls Trackerdaten hï¿½heren
      Wert aufweisen
00124         speed = MAX_SPEED;
00125
00126     return speed; //Zuweisung der Geschwindigkeit
00127 }
00128
00129 void CRobCodeGenerator::calculateAngles(COutputPoint3D& p, CInputPoint3D& pIn)
00130 {
00131     // Funktion in Eulermatrix aufrufen die a/b/c neu berechnet
00132
00133     CEulerMatrix matrix = pIn.getEulerMatrix();
00134     tuple<double, double, double> abc;
00135
00136     abc = matrix.calculateAngels();
00137
00138     p.setA(get<0>(abc));
00139     p.setB(get<1>(abc));
00140     p.setC(get<2>(abc));
00141 }
```

6.39 src/RobPathEditor.cpp-Dateireferenz

: Hier wird die main Funktion aufgerufen

```
#include ¨./header/SegmentApproximator.h¨
#include ¨./header/PathBuilder.h¨
#include ¨./header/RobCodeGenerator.h¨
#include ¨./header/InputParameter.h¨
#include ¨./header/MeanFilter.h¨
#include ¨./header/GUI.h¨
#include ¨./header/Logging.h¨
#include <iostream>
#include <ctime>
```

Funktionen

- int main ()

6.39.1 Ausführliche Beschreibung

: Hier wird die main Funktion aufgerufen

Definiert in Datei RobPathEditor.cpp.

6.39.2 Dokumentation der Funktionen

6.39.2.1 main()

```
int main ( )
```

Definiert in Zeile 55 der Datei RobPathEditor.cpp.

```
00056 {
00057     clock_t start;
00058     start = clock();
00059
00060     try
00061     {
00062         //logging Initialisieren
00063         string loggingPath = "output";
00064         CLogging logging(loggingPath);
00065
00066         //read Data
00067
00068         CInputParameter inputParameter;
00069         string path = "input/path_01.csv";
00070         inputParameter.openFile(path);
00071
00072         //moving Average
00073
00074         CMeanFilter meanFilter;
00075         meanFilter.setWindowSize(3);
00076         meanFilter.mean(inputParameter.getPath(), logging);
00077
00078         // Douglas-Peuker Algorithm
00079
00080         CSegmentApproximator segmentApproximator;
00081         segmentApproximator.setmaxDistance(0.5);
00082         segmentApproximator.approx(meanFilter.getPath(), logging);
00083
00084         // Puts the Segments together to one path
00085
00086         CPathBuilder pathBuilder;
00087         pathBuilder.createPath(segmentApproximator.getSegmentsApproxVector(), logging);
00088
00089         // Calculates Speed, Angle and generates the Output Data
00090
00091         CRobCodeGenerator codeGenerator(inputParameter.getSpeed(), inputParameter.getSpeedManual(),
00092             inputParameter.getOrientationManual(), inputParameter.getAngles());
00093         codeGenerator.generateRobCode(pathBuilder.getPath(), loggingPath + "/ robCode.src");
00094
00095         float elapsed = (float)(clock() - start) / CLOCKS_PER_SEC;
00096     }
00097
00098     catch (exception& e)
00099     {
00100         cerr << e.what() << "\n";
00101     }
00102
00103     system("pause");
00104
00105    return 0;
00106 }
```

6.40 RobPathEditor.cpp

gehe zur Dokumentation dieser Datei

```
00001
00043 #include "./header/SegmentApproximator.h"
00044 #include "./header/PathBuilder.h"
00045 #include "./header/RobCodeGenerator.h"
00046 #include "./header/InputParameter.h"
00047 #include "./header/MeanFilter.h"
00048 #include "./header/GUI.h"
00049 #include "./header/Logging.h"
00050 #include <iostream>
00051 #include <ctime>
00052
00053 using namespace std;
00054
00055 int main()
00056 {
```

```
00057     clock_t start;
00058     start = clock();
00059
00060     try
00061     {
00062         //logging Initialisieren
00063         string loggingPath = "output";
00064         CLogging logging(loggingPath);
00065
00066         //read Data
00067
00068         CInputParameter inputParameter;
00069         string path = "input/path_01.csv";
00070         inputParameter.openFile(path);
00071
00072         //moving Average
00073
00074         CMeanFilter meanFilter;
00075         meanFilter.setWindowSize(3);
00076         meanFilter.mean(inputParameter.getPath(), logging);
00077
00078         // Douglas-Peuker Algorithm
00079
00080         CSegmentApproximator segmentApproximator;
00081         segmentApproximator.setmaxDistance(0.5);
00082         segmentApproximator.approx(meanFilter.getPath(), logging);
00083
00084         // Puts the Segments together to one path
00085
00086         CPathBuilder pathBuilder;
00087         pathBuilder.createPath(segmentApproximator.getSegmentsApproxVector(), logging);
00088
00089         // Calculates Speed, Angle and generates the Output Data
00090
00091         CRobCodeGenerator codeGenerator(inputParameter.getSpeed(), inputParameter.getSpeedManual(),
00092             inputParameter.getOrientationManual(), inputParameter.getAngles());
00093         codeGenerator.generateRobCode(pathBuilder.getPath(), loggingPath + "/ robCode.src");
00094
00095         float elapsed = (float)(clock() - start) / CLOCKS_PER_SEC;
00096     }
00097
00098     catch (exception& e)
00099     {
00100         cerr « e.what() « "\n";
00101     }
00102
00103     system("pause");
00104
00105    return 0;
00106 }
```

6.41 src/SegmentApproximator.cpp-Dateireferenz

: Source File Douglas-Peuker

```
#include ¨./header/SegmentApproximator.h¨
#include ¨./header/Point3D.h¨
#include ¨./header/Line3D.h¨
```

6.41.1 Ausführliche Beschreibung

: Source File Douglas-Peuker

Definiert in Datei SegmentApproximator.cpp.

6.42 SegmentApproximator.cpp

gehe zur Dokumentation dieser Datei

```
00001
00007 #include "./header/SegmentApproximator.h"
00008 #include "./header/Point3D.h"
00009 #include "./header/Line3D.h"
00010
00011 CSegmentApproximator::CSegmentApproximator(void)
00012 {
00013 }
00014
00015 CSegmentApproximator::~CSegmentApproximator(void)
00016 {
00017 }
00018
00019 void CSegmentApproximator::approx(const vector<list<CInputPoint3D»& segments, CLogging log)
00020 {
00021     CInputPoint3D p;
00022
00023     segmentsApprox = segments;
00024
00025     for (size_t s = 0; s < segments.size(); s++)
00026     {
00027         douglasPeuckerRecursive(segmentsApprox[s], segmentsApprox[s].begin(),
      --(segmentsApprox[s].end()));
00028     }
00029     log.setStep(3);
00030     log.logData(segmentsApprox);
00031 }
00032
00033 void CSegmentApproximator::setmaxDistance(double maxDistanceSource)
00034 {
00035     maxDistance = maxDistanceSource;
00036 }
00037
00038 double CSegmentApproximator::getmaxDistance()
00039 {
00040     return maxDistance;
00041 }
00042
00043 vector<list<CInputPoint3D»& CSegmentApproximator::getSegmentsApproxVector()
00044 {
00045     return segmentsApprox;
00046 }
00047
00048 //TODO: Kommentar
00049 void CSegmentApproximator::douglasPeuckerRecursive(list<CInputPoint3D>& segment,
      std::list<CInputPoint3D>::iterator startItr, std::list<CInputPoint3D>::iterator endItr)
00050 {
00051     if (segment.size() < 3) return;
00052     if (distance(startItr, endItr) == 2) return;
00053     CInputPoint3D pStart; CInputPoint3D pEnd;
00054     pStart.setX(startItr->getX()); pStart.setY(startItr->getY()); pStart.setZ(startItr->getZ());
00055
00056     pEnd.setX(endItr->getX()); pEnd.setY(endItr->getY()); pEnd.setZ(endItr->getZ());
00057
00058     double dist = 0.0, maxDist = 0.0;
00059     std::list<CInputPoint3D>::iterator maxItr, itr;
00060
00061     for (itr = startItr; itr != endItr; itr++)
00062     {
00063         CLine3D line = CLine3D(pStart, pEnd);
00064         // calc distance
00065         dist = itr->distanceTo(line);
00066         if (dist > maxDist) {
00067             maxDist = dist;
00068             maxItr = itr;
00069         }
00070     }
00071
00072     if (maxDist <= maxDistance) {
00073
00074         segment.erase((++startItr), endItr);
00075         return;
00076     }
00077
00078     douglasPeuckerRecursive(segment, startItr, maxItr);
00079     douglasPeuckerRecursive(segment, maxItr, endItr);
00080 }
```

Index

~CEulerMatrix
CEulerMatrix, 10
~CGUI
CGUI, 14
~CInputParameter
CInputParameter, 16
~CInputPoint3D
CInputPoint3D, 24
~CLine3D
CLine3D, 27
~CLogging
CLogging, 29
~CMeanFilter
CMeanFilter, 34
~COutputPoint3D
COutputPoint3D, 39
~CPathBuilder
CPathBuilder, 45
~CPoint3D
CPoint3D, 48
~CRobCodeGenerator
CRobCodeGenerator, 54
~CSegmentApproximator
CSegmentApproximator, 59

A
CInputParameter, 20
CRobCodeGenerator, 57
a
COutputPoint3D, 43
angels2mat
CEulerMatrix, 11
approx
CSegmentApproximator, 60

B
CInputParameter, 20
CRobCodeGenerator, 57
b
COutputPoint3D, 43
Beschreibung Roboter Path Editor, 1

C
CInputParameter, 20
CRobCodeGenerator, 57
c
COutputPoint3D, 43
calculateAngels
CEulerMatrix, 11
calculateAngles
CRobCodeGenerator, 54
calculateMean
CMeanFilter, 34
calculateSpeed
CRobCodeGenerator, 55
CEulerMatrix, 9
~CEulerMatrix, 10
angels2mat, 11
calculateAngels, 11
CEulerMatrix, 10
eulerMatrix, 13
getEulerMatrix, 12
getMatrix, 12
setMatrix, 12
CGUI, 13
~CGUI, 14
CGUI, 14
CInputParameter, 14
~CInputParameter, 16
A, 20
B, 20
C, 20
CInputParameter, 15
detectJump, 16
difference, 20
getAngles, 17
getOrientationManual, 17
getPath, 17
getSpeed, 18
getSpeedManual, 18
initialPath, 21
openFile, 18
orientationManual, 21
setOrientation, 19
setSpeed, 20
speed, 21
speedManual, 21
CInputPoint3D, 22
~CInputPoint3D, 24
CInputPoint3D, 23
getEulerMatrix, 24
getTime, 24
orientationMatrix, 26
setEulerMatrix, 25
setPoint, 25
setTime, 26
timestamp, 26
CLine3D, 26
~CLine3D, 27

CLine3D, 27
p1, 28
p2, 28
CLogging, 28
~CLogging, 29
CLogging, 29
logData, 30
path, 32
setStep, 31
step, 32
CMeanFilter, 32
~CMeanFilter, 34
calculateMean, 34
CMeanFilter, 33
getPath, 35
getWindowSize, 35
mean, 35
meanPath, 36
setWindowSize, 36
windowSize, 36
COutputPoint3D, 37
~COutputPoint3D, 39
a, 43
b, 43
c, 43
COutputPoint3D, 38
getA, 40
getB, 40
getC, 40
getSpeed, 40
setA, 41
setB, 41
setC, 41
setSpeed, 43
speed, 43
CPathBuilder, 44
~CPathBuilder, 45
CPathBuilder, 45
createPath, 45
getPath, 46
path, 46
CPoint3D, 46
~CPoint3D, 48
CPoint3D, 48
distanceTo, 49
getX, 50
getY, 50
getZ, 50
set, 50
setX, 51
setY, 51
setZ, 51
x, 52
y, 52
z, 52
createPath
CPathBuilder, 45
CRobCodeGenerator, 52
~CRobCodeGenerator, 54
A, 57
B, 57
C, 57
calculateAngles, 54
calculateSpeed, 55
CRobCodeGenerator, 53
generateRobCode, 55
orientationManual, 57
postProcessing, 56
processedPath, 58
speed, 58
speedManual, 58
CSegmentApproximator, 58
~CSegmentApproximator, 59
approx, 60
CSegmentApproximator, 59
douglasPeuckerRecursive, 60
getmaxDistance, 61
getSegmentsApproxVector, 61
maxDistance, 62
segmentsApprox, 62
setmaxDistance, 61

detectJump
CInputParameter, 16
difference
CInputParameter, 20
distanceTo
CPoint3D, 49
douglasPeuckerRecursive
CSegmentApproximator, 60

eulerMatrix
CEulerMatrix, 13

generateRobCode
CRobCodeGenerator, 55
getA
COutputPoint3D, 40
getAngles
CInputParameter, 17
getB
COutputPoint3D, 40
getC
COutputPoint3D, 40
getEulerMatrix
CEulerMatrix, 12
CInputPoint3D, 24
getMatrix
CEulerMatrix, 12
getmaxDistance
CSegmentApproximator, 61
getOrientationManual
CInputParameter, 17
getPath
CInputParameter, 17
CMeanFilter, 35
CPathBuilder, 46

getSegmentsApproxVector
CSegmentApproximator, 61
getSpeed
CInputParameter, 18
COutputPoint3D, 40
getSpeedManual
CInputParameter, 18
getTime
CInputPoint3D, 24
getWindowSize
CMeanFilter, 35
getX
CPoint3D, 50
getY
CPoint3D, 50
getZ
CPoint3D, 50

header/EulerMatrix.h, 63
header/GUI.h, 64
header/InputParameter.h, 64, 65
header/Line3D.h, 65, 66
header/Logging.h, 66, 67
header/MeanFilter.h, 67, 68
header/PathBuilder.h, 68, 69
header/Point3D.h, 69, 70
header/RobCodeGenerator.h, 70, 71
header/SegmentApproximator.h, 72

initialPath
CInputParameter, 21

logData
CLogging, 30

main
RobPathEditor.cpp, 87
MAX_SPEED
RobCodeGenerator.h, 71
maxDistance
CSegmentApproximator, 62
mean
CMeanFilter, 35
meanPath
CMeanFilter, 36

openFile
CInputParameter, 18
orientationManual
CInputParameter, 21
CRobCodeGenerator, 57
orientationMatrix
CInputPoint3D, 26

p1
CLine3D, 28
p2
CLine3D, 28
path
CLogging, 32
CPathBuilder, 46
postProcessing
CRobCodeGenerator, 56
processedPath
CRobCodeGenerator, 58

RobCodeGenerator.h
MAX_SPEED, 71
RobPathEditor.cpp
main, 87

segmentsApprox
CSegmentApproximator, 62
set
CPoint3D, 50
setA
COutputPoint3D, 41
setB
COutputPoint3D, 41
setC
COutputPoint3D, 41
setEulerMatrix
CInputPoint3D, 25
setMatrix
CEulerMatrix, 12
setmaxDistance
CSegmentApproximator, 61
setOrientation
CInputParameter, 19
setPoint
CInputPoint3D, 25
setSpeed
CInputParameter, 20
COutputPoint3D, 43
setStep
CLogging, 31
setTime
CInputPoint3D, 26
setWindowSize
CMeanFilter, 36
setX
CPoint3D, 51
setY
CPoint3D, 51
setZ
CPoint3D, 51
speed
CInputParameter, 21
COutputPoint3D, 43
CRobCodeGenerator, 58
speedManual
CInputParameter, 21
CRobCodeGenerator, 58
src/EulerMatrix.cpp, 73
src/GUI.cpp, 74
src/InputParameter.cpp, 74, 75
src/Line3D.cpp, 76, 77
src/Logging.cpp, 77
src/MeanFilter.cpp, 79

src/PathBuilder.cpp, 80
src/Point3D.cpp, 81
src/RobCodeGenerator.cpp, 84
src/RobPathEditor.cpp, 86, 87
src/SegmentApproximator.cpp, 88, 89
step
    CLogging, 32

timestamp
    CInputPoint3D, 26

windowSize
    CMeanFilter, 36

x
    CPoint3D, 52

y
    CPoint3D, 52

z
    CPoint3D, 52