RobCodeGenerator

Erzeugt von Doxygen 1.9.7

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:

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

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

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 5

Klassen-Dokumentation

5.1 CEulerMatrix Klassenreferenz

Handling und Berechnung Euler Matrix.

`#include <EulerMatrix.h>`

Öffentliche 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 Ausführliche Beschreibung

Handling und Berechnung Euler Matrix.

Diese Klasse speichert die Euler Matrix und hat Funktionen fï¿½r Berechnungen mit eben jener.

Definiert in Zeile 20 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 }
```

Benutzt eulerMatrix.

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 }
```

Benutzt eulerMatrix.

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

Rückgabe

: float inputMatrix[3][3] gibt die neu berechnete Matrix zurï¿½ck

Definiert in Zeile 65 der Datei EulerMatrix.cpp.

```
00066 {
00067     float Matrix[3][3];     // DummyMatrix erstellen
00068
00069     /* Berechnung der Matrix */
00070
00071     Matrix[0][0] = cos(A) * cos(C) - sin(A) * cos(B) * sin(C);
00072     Matrix[0][1] = -cos(A) * sin(C) - sin(A) * cos(B) * cos(C);
00073     Matrix[0][2] = sin(A) * sin(B);
00074
00075     Matrix[1][0] = sin(A) * cos(C) + cos(A) * cos(B) * sin(C);
00076     Matrix[1][1] = -sin(A) * sin(C) + cos(A) * cos(B) * cos(C);
00077     Matrix[1][2] = -cos(A) * sin(B);
00078
00079     Matrix[2][0] = sin(B) * sin(C);
00080     Matrix[2][1] = sin(B) * cos(C);
00081     Matrix[2][2] = cos(B);
00082
00083     CEulerMatrix buffer(Matrix);        // DummyMatrix in DummyEulerMatrix schreiben
00084     return buffer;                      // Matrix zurÃ¼ck geben
00085 }
```

5.1.3.2 calculateAngels()

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

Berechnet die Kuka Wunkel A,B,C.

Rückgabe

: tuple<double , double , double> gibt die berechneten Winkel A, B, C zurÃ¼ck

Definiert in Zeile 88 der Datei EulerMatrix.cpp.

```
00089 {
00090     double a, b, c, sin_a, cos_a, sin_b, abs_cos_b, sin_c, cos_c;
00091
00092     /*
00093     a == Winkel Alpha
00094     b == Winkel Beta
00095     c == Winkel Gamma
00096
00097     sin_a == sinus alpha
00098     cos_a == cosinus alpha
00099     sin_b == Matrix[2][0] * -1
00100     abs_cos_b == ??
00101     sin_c == sinus gamma
00102     cos_c == cosinus gamma
00103     */
00104
00105
00106     /* Berechnung von alpha*/
00107     a = atan2(eulerMatrix[1][0], eulerMatrix[0][0]);
00108
00109     /* Berechung von beta */
00110     sin_a = sin(a);
00111     cos_a = cos(a);
00112     sin_b = eulerMatrix[2][0] * -1;
00113     abs_cos_b = cos(a) * eulerMatrix[0][0] + sin(a) * eulerMatrix[1][0];
00114
00115     b = atan2 (sin_b, abs_cos_b);
00116
00117     /* Berechung von gamma */
00118     sin_c =  sin_a * eulerMatrix[0][2] - cos_a * eulerMatrix[1][2];
00119     cos_c = -sin_a * eulerMatrix[0][1] + cos_a * eulerMatrix[1][1];
00120
00121     c = atan2(sin_c, cos_c);
00122
00123     /* Bogenmass in Gradmass umrechnen */
00124     a = a * 180 / M_PI;
00125     b = b * 180 / M_PI;
00126     c = c * 180 / M_PI;
00127
00128
00129     return make_tuple(a, b, c);     // RÃ¼ckgabe der Winkel
00130 }
```

Benutzt eulerMatrix.

Wird benutzt von CRobCodeGenerator::calculateAngles().

5.1.3.3 getEulerMatrix()

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

Auslesen eine Matrix.

Rückgabe

: float inputMatrix[3][3] gibt gespeicherte Matrix zurï¿½ck

Definiert in Zeile 48 der Datei EulerMatrix.cpp.

```
00049 {
00050     return eulerMatrix;         // EulerMatrix zurÃ¼ck geben
00051 }
```

Benutzt eulerMatrix.

Wird benutzt von CLogging::logData() und CInputParameter::openFile().

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 ï¿½bergabewerten ï¿½berschreiben
00060         }
00061     }
00062 }
```

Benutzt eulerMatrix.

Wird benutzt von CLogging::logData() und CLogging::logData().

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 Ãbergabewerten Ã¼berschreiben
00044         }
00045     }
00046 }
```

Benutzt eulerMatrix.

Wird benutzt von CInputParameter::openFile().

5.1.4 Dokumentation der Datenelemente

5.1.4.1 eulerMatrix

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

Gespeicherte Euler Matrix

Definiert in Zeile 76 der Datei EulerMatrix.h.

Wird benutzt von calculateAngels(), CEulerMatrix(), CEulerMatrix(), getEulerMatrix(), getMatrix() und setMatrix().

Die Dokumentation für 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)

Definiert in Zeile 24 der Datei InputParameter.cpp.

```
00025 {
00026     speed = 0.1;
00027     A = 0;
00028     B = 75;
00029     C = 0;
00030     speedManual = true,
00031     orientationManual = true;
00032
00033 }
```

Benutzt A, B, C, orientationManual, speed und speedManual.

5.3.2.2 CInputParameter() [2/2]

```
CInputParameter::CInputParameter (
            double initSpeed,
            bool initSeepManual,
            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);

Definiert in Zeile 12 der Datei InputParameter.cpp.

```
00013 {
00014     speed = initSpeed;
00015     speedManual = initSpeedManual;
00016     orientationManual = initOrientationManual;
00017     A = initA;
00018     B = initB;
00019     C = initC;
00020
00021 }
```

Benutzt A, B, C, orientationManual, speed und speedManual.

5.3.2.3 ~CInputParameter()

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

Dekonstruktor.

Definiert in Zeile 35 der Datei InputParameter.cpp.

```
00036 {
00037
00038 }
```

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 126 der Datei InputParameter.cpp.

```
00127 {
00128     if(abs(p.getX() - x_prev) > difference)              // Abstand zwischen Punkten größer max
      Differenz??
00129         return true;
00130     else if(abs(p.getY() - y_prev) > difference)      // Abstand zwischen Punkten größer max
      Differenz??
00131         return true;
```

```
00132     else if(abs(p.getZ() - z_prev) > difference)          // Abstand zwischen Punkten größer max
      Differenz??
00133         return true;
00134     else
00135         return false;
00136 }
```

Benutzt difference, CPoint3D::getX(), CPoint3D::getY() und CPoint3D::getZ().

Wird benutzt von openFile().

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 77 der Datei InputParameter.cpp.

```
00078 {
00079     return make_tuple(A, B, C);      // Winkel zurück geben
00080 }
```

Benutzt A, B und C.

Wird benutzt von main().

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 72 der Datei InputParameter.cpp.

```
00073 {
00074     return orientationManual;      // Vorgewählte Einstellung für Orientierung zurück geben
00075 }
```

Benutzt orientationManual.

Wird benutzt von main().

5.3.3.4 getPath()

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

Gibt Pfad zurück.

Rückgabe

: vector<list<CInputPoint3D>> den eingelesen Pfad

Definiert in Zeile 57 der Datei InputParameter.cpp.

```
00058 {
00059     return initialPath;      // Path zurück geben
00060 }
```

Benutzt initialPath.

Wird benutzt von main().

5.3.3.5 getSpeed()

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

Gibt Geschwindigkeit zurück.

Gibt die eingegebene Geschwindigkeit zurück

Definiert in Zeile 62 der Datei InputParameter.cpp.

```
00063 {
00064     return speed;          // Geschwindigkeit zurück geben
00065 }
```

Benutzt speed.

Wird benutzt von main().

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 67 der Datei InputParameter.cpp.

```
00068 {
00069     return speedManual;             // Vorgewählte Einstellung für Geschwindigkeit zurück geben
00070 }
```

Benutzt speedManual.

Wird benutzt von main().

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 84 der Datei InputParameter.cpp.

```
00085 {
00086     ifstream fin(path);
00087     CInputPoint3D tmpPoint;        // Zwischenspeicher zum konvertieren von tmpEuler in Point3D
00088     CEulerMatrix tmpEuler;      // Zwischenspeicher zum konverteiren von DummyMatrix in EulerMatrix
00089     double x, y, z;             // Punktkoordinaten
00090     double x_prev = 0, y_prev = 0, z_prev = 0;      // Zwischenspeicher für Punktkoordinaten
00091     double timestamp;           // Zeitstempel
00092     int segmentCount = -1;      // Segmentzähler
00093     float dummyMatrix[3][3];    // DummyMatrix zum speichern
00094
00095
00096     if (!fin.is_open())
00097     {
00098         cerr « "Datei konnte nicht geöffnet werden" « endl;       // Fehler Datei konnte nicht
      geöffnet werden.
00099     }
00100     string line;
00101
00102     while(getline(fin, line))
00103     {
00104         std::istringstream sStream (line);
00105         sStream » timestamp » x » y » z » dummyMatrix[0][0] » dummyMatrix[0][1] » dummyMatrix[0][2]
      // Zeile in die einzelnen Parameter zerlegen
00106             » dummyMatrix[1][0] » dummyMatrix[1][1] » dummyMatrix[1][2] » dummyMatrix[2][0] »
      dummyMatrix[2][1] » dummyMatrix[2][2];      // und in DummyMatrix bzw. Variablen abspeichern
00107
00108         tmpEuler.setMatrix(dummyMatrix);                                    // DummyMatrix[3][3] in
      EulerMatrix speichern
00109         tmpPoint.setPoint(timestamp, x, y, z, tmpEuler.getEulerMatrix()); // Variablen und EulerWinkel
      in CPoint3D speichern
00110
00111         if (detectJump(tmpPoint, x_prev, y_prev, z_prev)) // if there is a jump in the data, start da
      new segment
00112         {
00113             segmentCount++;                                 // neues Segement anlegen
00114             initialPath.push_back(list<CInputPoint3D>());    // Punkt in Segent speichern
00115         }
00116
00117         initialPath[segmentCount].push_back(tmpPoint);       // Punkt in bestehendes Segment
      abspeichern
00118
00119         x_prev = x;     // X-Wert zwischenspeichern
00120         y_prev = y;     // Y-Wert zwischenspeichern
00121         z_prev = z;     // Z-Wert zwischenspeichern
00122     }
00123     fin.close();        // Datei schließen
00124 }
```

Benutzt detectJump(), CEulerMatrix::getEulerMatrix(), initialPath, CEulerMatrix::setMatrix() und CInputPoint3D::setPoint().

Wird benutzt von main().

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 42 der Datei InputParameter.cpp.

```
00043 {
00044     orientationManual = initOrientationManual;
00045     A = initA;
00046     B = initB;
00047     C = initC;
00048 }
```

Benutzt A, B, C und orientationManual.

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 51 der Datei InputParameter.cpp.

```
00052 {
00053     speed = initSpeed;
00054     speedManual = initSpeedManual;
00055 }
```

Benutzt speed und speedManual.

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.

Wird benutzt von CInputParameter(), CInputParameter(), getAngles() und setOrientation().

5.3.4.2 B

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

User eingegebener Winkel B

Definiert in Zeile 137 der Datei InputParameter.h.

Wird benutzt von CInputParameter(), CInputParameter(), getAngles() und setOrientation().

5.3.4.3 C

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

User eingegebener Winkel C

Definiert in Zeile 141 der Datei InputParameter.h.

Wird benutzt von CInputParameter(), CInputParameter(), getAngles() und setOrientation().

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.

Wird benutzt von detectJump().

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.

Wird benutzt von getPath() und openFile().

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.

Wird benutzt von CInputParameter(), CInputParameter(), getOrientationManual() und setOrientation().

5.3.4.7 speed

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

User eingegebene Geschwindigkeit

Definiert in Zeile 121 der Datei InputParameter.h.

Wird benutzt von CInputParameter(), CInputParameter(), getSpeed() und setSpeed().

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.

Wird benutzt von CInputParameter(), CInputParameter(), getSpeedManual() und setSpeed().

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

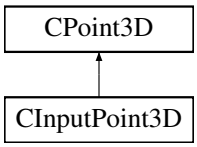
- header/InputParameter.h
- src/InputParameter.cpp

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 110 der Datei Point3D.cpp.

```
00110                                   : CPoint3D()
00111 {
00112     timestamp = 0;     // Zeitstempel mit 0 initialisieren
00113 }
```

Benutzt timestamp.

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 116 der Datei Point3D.cpp.

```
00117 {
00118     x = X;
00119     y = Y;
00120     z = Z;
00121     timestamp = Timestamp;
00122     orientationMatrix = Matrix;
00123 }
```

Benutzt orientationMatrix, timestamp, CPoint3D::x, CPoint3D::y und CPoint3D::z.

5.4.2.3 ~CInputPoint3D()

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

Dekonstruktor.

Definiert in Zeile 125 der Datei Point3D.cpp.

```
00126 {
00127 }
```

5.4.3 Dokumentation der Elementfunktionen

5.4.3.1 getEulerMatrix()

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

Gibt die gespeicherte Eulermatrix zurück.

Rückgabe

CEulerMatrix

Definiert in Zeile 147 der Datei Point3D.cpp.

```
00148 {
00149     return orientationMatrix;  // Rückgabe der EulerMatrix
00150 }
```

Benutzt orientationMatrix.

Wird benutzt von CRobCodeGenerator::calculateAngles().

5.4.3.2 getTime()

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

Gibt den Zeitstempel zurück.

Rückgabe

double Zeitstempel

Definiert in Zeile 152 der Datei Point3D.cpp.

```
00153 {
00154     return timestamp;      // Rückgabe des Zeitstempel
00155 }
```

Benutzt timestamp.

Wird benutzt von CRobCodeGenerator::calculateSpeed().

5.4.3.3 setEulerMatrix()

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

Setzt die Eulermatrix.

Parameter

CEulerMatrix	orientation

Definiert in Zeile 129 der Datei Point3D.cpp.

```
00130 {
00131     orientationMatrix = orientation;   // EulerMatrix setzen
00132 }
```

Benutzt orientationMatrix.

Wird benutzt von CMeanFilter::calculateMean(), CPathBuilder::createPath(), CSegmentApproximator::douglasPeuckerRecursive() und setPoint().

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 135 der Datei Point3D.cpp.

```
00136 {
00137     setTime(time);    // Zeitstempel setzen
00138     set(X, Y, Z);     // setze Punkt-Koordinaten
00139     setEulerMatrix(orientation); // EulerMatrix setzen
00140 }
```

Benutzt CPoint3D::set(), setEulerMatrix() und setTime().

Wird benutzt von CInputParameter::openFile().

5.4.3.5 setTime()

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

Setzt den Zeitstempel.

Parameter

double	time

Definiert in Zeile 142 der Datei Point3D.cpp.

```
00143 {
00144     timestamp = time;  // Zeitstempel setzen
00145 }
```

Benutzt timestamp.

Wird benutzt von CMeanFilter::calculateMean(), CPathBuilder::createPath(), CSegmentApproximator::douglasPeuckerRecursive() und setPoint().

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.

Wird benutzt von CInputPoint3D(), getEulerMatrix() und setEulerMatrix().

5.4.4.2 timestamp

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

Zeitstempel

Definiert in Zeile 166 der Datei Point3D.h.

Wird benutzt von CInputPoint3D(), CInputPoint3D(), getTime() und setTime().

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 15 der Datei Line3D.cpp.

```
00016 {
00017     p1 = P1;
00018     p2 = P2;
00019 }
```

Benutzt p1 und p2.

5.5.2.3 ~CLine3D()

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

Dekonstruktor.

Definiert in Zeile 21 der Datei Line3D.cpp.

```
00022 {
00023 }
```

5.5.3 Dokumentation der Datenelemente

5.5.3.1 p1

```
CPoint3D CLine3D::p1
```

Punkt 1

Definiert in Zeile 41 der Datei Line3D.h.

Wird benutzt von CLine3D() und CPoint3D::distanceTo().

5.5.3.2 p2

```
CPoint3D CLine3D::p2
```

Punkt 2

Definiert in Zeile 45 der Datei Line3D.h.

Wird benutzt von CLine3D() und CPoint3D::distanceTo().

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 10 der Datei Logging.cpp.

```
00011 {
00012     step = 0;
00013 }
```

Benutzt step.

5.6.2.2 CLogging() [2/2]

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

Default Konstruktor.

Initialisiert Logging Klasse

Siehe auch

CMeanFilter(int Window);

Definiert in Zeile 16 der Datei Logging.cpp.

```
00017 {
00018     path = Path;
00019 }
```

Benutzt path.

5.6.2.3 ~CLogging()

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

Dekonstruktor.

Definiert in Zeile 21 der Datei Logging.cpp.

```
00022 {
00023
00024 }
```

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 67 der Datei Logging.cpp.

```
00068 {
00069     string filepath;            // file Pfad
00070     float dummyMatrix[3][3];    // dummyMatrix zum Zwischenspeichern
00071     CEulerMatrix tmpEuler;      // CEulerMatrix zum Zwischenspeichern
00072
00073     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
```

```
00074
00075     FILE* fid = fopen(filepath.c_str(), "w");   // file öffnen
00076
00077     if (fid == NULL)
00078     {
00079         cerr « "ERROR - Can NOT write to output file!\n";  // Fehler beim file öffnen
00080         return;
00081     }
00082
00083     /* Ausgeben der Punkte mit dummyMatrix */
00084     for (size_t s = 0; s < sourcePath.size(); s++) //for all points in the vector
00085     {
00086         tmpEuler.getMatrix(dummyMatrix);
00087
00088         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f\n", (double)sourcePath[s].getTime(),
00089             (double)sourcePath[s].getX(), (double)sourcePath[s].getY(), (double)sourcePath[s].getZ(),
00090             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2],
00091             dummyMatrix[1][0], dummyMatrix[1][1], dummyMatrix[1][2],
00092             dummyMatrix[2][0], dummyMatrix[2][1], dummyMatrix[2][2]);
00093     }
00094 }
```

Benutzt CEulerMatrix::getMatrix(), path und step.

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 31 der Datei Logging.cpp.

```
00032 {
00033     string filepath;            // file Pfad
00034     float dummyMatrix[3][3];    // dummyMatrix zum Zwischenspeichern
00035     CEulerMatrix tmpEuler;      // CEulerMatrix zum Zwischenspeichern
00036
00037     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00038
00039     FILE* fid = fopen(filepath.c_str(), "w");       // file öffnen
00040
00041     if (fid == NULL)
00042     {
00043         cerr « "ERROR - Can NOT write to output file!\n";      // Fehler beim file öffnen
00044         return;
00045     }
00046
00047     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00048     {
00049         list<CInputPoint3D>::iterator itr = sourcePath[s].begin();
00050
00051         tmpEuler = itr->getEulerMatrix();
00052         tmpEuler.getMatrix(dummyMatrix);
00053
00054         /* Ausgeben der Punkte mit dummyMatrix */
00055         for (; itr != sourcePath[s].end(); itr++) //for all points in the segment
00056         {
00057             fprintf(fid, "%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],
00059                 dummyMatrix[1][0], dummyMatrix[1][1], dummyMatrix[1][2],
00060                 dummyMatrix[2][0], dummyMatrix[2][1], dummyMatrix[2][2]);
00061         }
00062
00063         itr--;
00064     }
00065 }
```

Benutzt CEulerMatrix::getEulerMatrix(), CEulerMatrix::getMatrix(), path und step.

Wird benutzt von CSegmentApproximator::approx(), CPathBuilder::createPath() und CMeanFilter::mean().

5.6.3.3 setStep()

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

Setzt in welchem Schritt wir uns befinden.

Parameter

int	Step

Definiert in Zeile 26 der Datei Logging.cpp.

```
00027 {
00028     step = Step;    // Step setzen
00029 }
```

Benutzt step.

Wird benutzt von CSegmentApproximator::approx(), CPathBuilder::createPath() und CMeanFilter::mean().

5.6.4 Dokumentation der Datenelemente

5.6.4.1 path

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

Speicherpfad

Definiert in Zeile 66 der Datei Logging.h.

Wird benutzt von CLogging(), logData() und logData().

5.6.4.2 step

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

In welchem Schritt sind wir gerade

Definiert in Zeile 62 der Datei Logging.h.

Wird benutzt von CLogging(), logData(), logData() und setStep().

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;           // initialisieren mit Standardfenstergröße 3
00014 }
```

Benutzt windowSize.

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 @seeCMeanFilter();

Definiert in Zeile 16 der Datei MeanFilter.cpp.

```
00017 {
00018     windowSize = Window;  // initialisieren der Fenstergröße mit Übergabewert
00019 }
```

Benutzt windowSize.

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;     // Variablen zum Speichern der Summe
00055     double div = 0;                          // Variable zum Speichern des Teilers
00056
00057     CInputPoint3D p;       //Point3D zum Zwischenspeichern
00058
00059     size_t inputSize = segment.size();
00060
00061     list<CInputPoint3D>::iterator it = segment.begin();
00062     list<CInputPoint3D> newSegment;
00063
00064     for (size_t i = 0; i < inputSize - windowSize; ++i) //For each element in the Segment
00065     {
00066         sumX = 0, sumY = 0, sumZ = 0;    // Variablen zum Speichern der Summe auf 0 zurück setzen
00067         div = 0;                         // Variable zum Speichern des Teilers auf 0 zurück setzen
00068         p.setTime(it->getTime());
00069         p.setEulerMatrix(it->getEulerMatrix());
00070         for (size_t j = i; j < i + windowSize; ++j) // Build the sums for the three points
00071         {
00072             sumX += it->getX();
00073             sumY += it->getY();
00074             sumZ += it->getZ();
00075             div++;
00076             it++;
00077         }
00078         for (size_t index = windowSize; index > 0; index--) // Pain, the iterator has to be set back
00079         {
00080             it--;
00081         }
00082         p.set(sumX / div, sumY / div, sumZ / div); // Calculate smoothed values
00083         if(it != segment.end())
00084             it++;
00085         newSegment.push_back(p);
00086     }
00087     return newSegment;
00088 }
```

Benutzt CPoint3D::set(), CInputPoint3D::setEulerMatrix(), CInputPoint3D::setTime() und windowSize.

Wird benutzt von mean().

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;        // Mittelwert zurück geben
00038 }
```

Benutzt meanPath.

Wird benutzt von main().

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;        // Fenstergröße zurück geben
00033 }
```

Benutzt windowSize.

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 }
```

Benutzt calculateMean(), CLogging::logData(), meanPath und CLogging::setStep().

Wird benutzt von main().

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;  // setzen der Fenstergröße mit Übergabewert
00028 }
```

Benutzt windowSize.

Wird benutzt von main().

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.

Wird benutzt von getPath() und mean().

5.7.4.2 windowSize

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

Grösse des Fensters des gleitenden Mittelwerts

Definiert in Zeile 79 der Datei MeanFilter.h.

Wird benutzt von calculateMean(), CMeanFilter(), CMeanFilter(), getWindowSize() und setWindowSize().

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

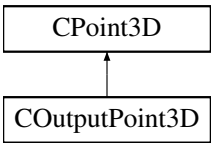
- header/MeanFilter.h
- src/MeanFilter.cpp

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 159 der Datei Point3D.cpp.

```
00159                                     : CPoint3D()
00160 {
00161     speed = 0;
00162     a = 0;
00163     b = 0;
00164     c = 0;
00165 }
```

Benutzt a, b, c und speed.

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 168 der Datei Point3D.cpp.

```
00169 {
00170     speed = Speed;
00171     a = A;
00172     b = B;
00173     c = C;
00174     x = X;
00175     y = Y;
00176     z = Z;
00177 }
```

Benutzt a, b, c, speed, CPoint3D::x, CPoint3D::y und CPoint3D::z.

5.8.2.3 ~COutputPoint3D()

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

Dekonstruktor.

Definiert in Zeile 179 der Datei Point3D.cpp.

```
00180 {
00181
00182 }
```

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 getA()

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

Gibt A zurück.

Rückgabe

: double A

Definiert in Zeile 184 der Datei Point3D.cpp.

```
00185 {
00186     return a;  // Rückgabe Winkel alpha
00187 }
```

Benutzt a.

5.8.3.2 getB()

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

Gibt B zurück.

Rückgabe

: double B

Definiert in Zeile 189 der Datei Point3D.cpp.

```
00190 {
00191     return b;  // Rückgabe Winkel beta
00192 }
```

Benutzt b.

5.8.3.3 getC()

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

Gibt C zurück.

Rückgabe

: double C

Definiert in Zeile 194 der Datei Point3D.cpp.

```
00195 {
00196     return c;  // Rückgabe Winkel gamma
00197 }
```

Benutzt c.

5.8.3.4 getSpeed()

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

Gibt die Geschwindigkeit zurück.

Rückgabe

: double Geschwindigkeit

Definiert in Zeile 199 der Datei Point3D.cpp.

```
00200 {
00201     return speed;  // Rückgabe Geschwindigkeit
00202 }
```

Benutzt speed.

5.8.3.5 setA()

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

Setzt A.

Parameter

double	A

Definiert in Zeile 204 der Datei Point3D.cpp.

```
00205 {
00206     a = A;     // setze Winkel alpha
00207 }
```

Benutzt a.

Wird benutzt von CRobCodeGenerator::calculateAngles() und CRobCodeGenerator::postProcessing().

5.8.3.6 setB()

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

Setzt B.

Parameter

double	B

Definiert in Zeile 209 der Datei Point3D.cpp.

```
00210 {
00211     b = B;     // setze Winkel beta
00212 }
```

Benutzt b.

Wird benutzt von CRobCodeGenerator::calculateAngles() und CRobCodeGenerator::postProcessing().

5.8.3.7 setC()

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

Setzt C.

Parameter

double	C

Definiert in Zeile 214 der Datei Point3D.cpp.

```
00215 {
00216     c = C;     // setze Winkel gamma
00217 }
```

Benutzt c.

Wird benutzt von CRobCodeGenerator::calculateAngles() und CRobCodeGenerator::postProcessing().

5.8.3.8 setSpeed()

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

Setzt die Geschwindigkeit.

Parameter

double	speed

Definiert in Zeile 219 der Datei Point3D.cpp.

```
00220 {
00221     speed = Speed;     // setze Geschwindigkeit
00222 }
```

Benutzt speed.

Wird benutzt von CRobCodeGenerator::postProcessing().

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.

Wird benutzt von COutputPoint3D(), COutputPoint3D(), getA() und setA().

5.8.4.2 b

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

Definiert in Zeile 249 der Datei Point3D.h.

Wird benutzt von COutputPoint3D(), COutputPoint3D(), getB() und setB().

5.8.4.3 c

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

Definiert in Zeile 249 der Datei Point3D.h.

Wird benutzt von COutputPoint3D(), COutputPoint3D(), getC() und setC().

5.8.4.4 speed

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

Geschwindigkeit zum Punkt hin? TODO: überprüfen

Definiert in Zeile 253 der Datei Point3D.h.

Wird benutzt von COutputPoint3D(), COutputPoint3D(), getSpeed() und setSpeed().

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
00027     for (size_t s = 0; s < segments.size(); s++) //for all segments
00028     {
00029         list<CInputPoint3D>::iterator itr = segments[s].begin();
00030
00031         for (; itr != segments[s].end(); itr++) //for all points in the segment
00032         {
00033             point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ());
00034             point.setTime(itr->getTime());
00035             point.setEulerMatrix(itr->getEulerMatrix());
00036             path.push_back(point);
00037         }
00038
00039         itr--;
00040     }
00041
00042     log.setStep(4);
00043     log.logData(path);
00044 }
```

Benutzt CLogging::logData(), path, CPoint3D::set(), CInputPoint3D::setEulerMatrix(), CLogging::setStep() und CInputPoint3D::setTime().

Wird benutzt von main().

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 }
```

Benutzt path.

Wird benutzt von main().

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.

Wird benutzt von createPath() und getPath().

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

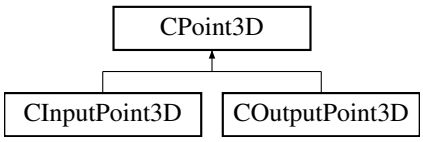
- header/PathBuilder.h
- src/PathBuilder.cpp

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 }
```

Benutzt x, y und z.

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 21 der Datei Point3D.cpp.

```
00022 {
00023     x = X;
00024     y = Y;
00025     z = Z;
00026 }
```

Benutzt x, y und z.

5.10.2.3 ~CPoint3D()

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

Dekonstruktor.

Definiert in Zeile 28 der Datei Point3D.cpp.

```
00029 {
00030 }
```

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 76 der Datei Point3D.cpp.

```
00077 {
00078     double bx, by, bz, rv_sq, dist, vp1, vp2, vp3;          // Variablen Anlegen
00079
00080     /*
00081     Vermessen wird der Punkt selbst
00082
00083     bx, by, bz      == Vektordifferenz
00084     rv_sq           == Betrag des Linienvektors
00085     dist            == Distanz von Punkt zu Linie
00086     vp1, vp2, vp3   == Vektorprodukte
00087     */
00088
00089     int rvx = line.p1.x - line.p2.x;      // Parameter X des Linienvektor berechnen
00090     int rvy = line.p1.y - line.p2.y;      // Parameter Y des Linienvektor berechnen
00091     int rvz = line.p1.z - line.p2.z;      // Parameter Z des Linienvektor berechnen
00092
00093     rv_sq = sqrt(((double)rvx * (double)rvx) + ((double)rvy * (double)rvy) + ((double)rvz *
      (double)rvz));      // Betrag des Linienvektor berechnen
00094
00095     bx = x - (double)line.p1.x;             // X(Punkt) - X(Aufpunkt)
00096     by = y - (double)line.p1.y;             // Y(Punkt) - Y(Aufpunkt)
00097     bz = z - (double)line.p1.z;             // Z(Punkt) - Z(Aufpunkt)
00098
00099     vp1 = by * rvz - bz * rvy;              // Parameter X Vektorprodukt
00100     vp2 = bz * rvx - bx * rvz;              // Parameter Y Vektorprodukt
00101     vp3 = bx * rvy - by * rvx;              // Parameter Z Vektorprodukt
00102
00103     dist = sqrt(vp1 * vp1 + vp2 * vp2 + vp3 * vp3) / rv_sq; // Betrag des Vektors berechnen
00104
00105     return dist;
00106 }
```

Benutzt CLine3D::p1, CLine3D::p2, x, y und z.

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 71 der Datei Point3D.cpp.

```
00072 {
00073     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
00074 }
```

Benutzt getX(), getY(), getZ(), x, y und z.

5.10.3.3 getX()

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

Gibt X zurück.

Rückgabe

double

Definiert in Zeile 32 der Datei Point3D.cpp.

```
00033 {
00034     return x;  // X-Koordinate zurück geben
00035 }
```

Benutzt x.

Wird benutzt von CInputParameter::detectJump() und distanceTo().

5.10.3.4 getY()

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

Gibt Y zurück.

Rückgabe

double

Definiert in Zeile 37 der Datei Point3D.cpp.

```
00038 {
00039     return y;  // Y-Koordinate zurück geben
00040 }
```

Benutzt y.

Wird benutzt von CInputParameter::detectJump() und distanceTo().

5.10.3.5 getZ()

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

Gibt Z zurück.

Rückgabe

double

Definiert in Zeile 42 der Datei Point3D.cpp.

```
00043 {
00044     return z;  // Z-Koordinate zurück geben
00045 }
```

Benutzt z.

Wird benutzt von CInputParameter::detectJump() und distanceTo().

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 63 der Datei Point3D.cpp.

```
00064 {
00065     x = X; // X-Koordinate setzen
00066     y = Y; // Y-Koordinate setzen
00067     z = Z; // Z-Koordinate setzen
00068 }
```

Benutzt x, y und z.

Wird benutzt von CMeanFilter::calculateMean(), CPathBuilder::createPath(), CRobCodeGenerator::postProcessing() und CInputPoint3D::setPoint().

5.10.3.7 setX()

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

Setzt X.

Parameter

double	X

Definiert in Zeile 47 der Datei Point3D.cpp.

```
00048 {
00049     x = X; // X-Koordinate setzen
00050 }
```

Benutzt x.

Wird benutzt von CSegmentApproximator::douglasPeuckerRecursive().

5.10.3.8 setY()

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

Setzt Y.

Parameter

double	Y

Definiert in Zeile 52 der Datei Point3D.cpp.

```
00053 {
00054     y = Y; // Y-Koordinate setzen
00055 }
```

Benutzt y.

Wird benutzt von CSegmentApproximator::douglasPeuckerRecursive().

5.10.3.9 setZ()

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

Setzt Z.

Parameter

double	Z

Definiert in Zeile 57 der Datei Point3D.cpp.

```
00058 {
00059     z = Z; // Z-Koordinate setzen
00060 }
```

Benutzt z.

Wird benutzt von CSegmentApproximator::douglasPeuckerRecursive().

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.

Wird benutzt von CInputPoint3D::CInputPoint3D(), COutputPoint3D::COutputPoint3D(), CPoint3D(), CPoint3D(), distanceTo(), distanceTo(), getX(), set() und setX().

5.10.4.2 y

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

Definiert in Zeile 99 der Datei Point3D.h.

Wird benutzt von CInputPoint3D::CInputPoint3D(), COutputPoint3D::COutputPoint3D(), CPoint3D(), CPoint3D(), distanceTo(), distanceTo(), getY(), set() und setY().

5.10.4.3 z

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

Definiert in Zeile 99 der Datei Point3D.h.

Wird benutzt von CInputPoint3D::CInputPoint3D(), COutputPoint3D::COutputPoint3D(), CPoint3D(), CPoint3D(), distanceTo(), distanceTo(), getZ(), set() und setZ().

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 filepath, 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 11 der Datei RobCodeGenerator.cpp.

```
00012 {
00013     speed = 0;
00014     speedManual = 0;
00015     orientationManual = 0;
00016     A = 0;
00017     B = 0;
00018     C = 0;
00019 }
```

Benutzt A, B, C, orientationManual, speed und speedManual.

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 22 der Datei RobCodeGenerator.cpp.

```
00023 {
00024     speed = Speed;
00025     speedManual = SpeedManual;
00026     orientationManual = OrientationManual;
00027     A = get<0>(angles);
00028     B = get<1>(angles);
00029     C = get<2>(angles);
00030 }
```

Benutzt A, B, C, orientationManual, speed und speedManual.

5.11.2.3 ~CRobCodeGenerator()

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

Dekonstruktor.

Definiert in Zeile 32 der Datei RobCodeGenerator.cpp.

```
00033 {
00034 }
```

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 }
```

Benutzt CEulerMatrix::calculateAngels(), CInputPoint3D::getEulerMatrix(), COutputPoint3D::setA(), COutputPoint3D::setB() und COutputPoint3D::setC().

Wird benutzt von postProcessing().

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

double

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 }
```

Benutzt CInputPoint3D::getTime(), MAX_SPEED, processedPath und speed.

Wird benutzt von postProcessing().

5.11.3.3 generateRobCode()

```
void CRobCodeGenerator::generateRobCode (
            vector< CInputPoint3D > & path,
            string filepath,
            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 36 der Datei RobCodeGenerator.cpp.

```
00037 {
00038     postProcessing(points); // Calculates all the necessary values
00039
00040     errno_t err;
00041
00042     FILE* fid;
00043
00044     string fullPath = filepath + "/" + filename;
00045
00046     if ((err = fopen_s(&fid, fullPath.c_str(), "w")) != 0) // Errorhandling for File opening
00047     {
00048         string msg = "Open file: ";
00049         msg += filename;
00050         msg += " failed!";
00051
00052         throw exception(msg.c_str());
00053     }
00054
00055     filename.erase(filename.end()-4,filename.end());        // lÃ¶scht .src
00056     fprintf(fid, "DEF %s \n", filename.c_str());            // DEF in file schreiben
00057
00058     fputs("PTP $POS_ACT\n", fid);                           // PTP zur aktuellen Position in file
      schreiben
00059
00060     if (speedManual) // If the speed is set to manual, it will be defined once at the beginning of the
      file
00061     {
00062         fprintf(fid, "$VEL.CP = %f\n", speed);     // Geschwindigkeit ein file schreiben
00063     }
00064
00065     for (size_t s = 0; s < points.size(); s++)
00066     {
00067         if (!speedManual) // If the speed is calculated it needs to be before every LIN command
00068             fprintf(fid, "&VEL.CP = %f\n", (float)processedPath[s].getSpeed());
00069         fprintf(fid, "LIN {X %f, Y %f, Z %f, A %f, B %f, C %f}\n", round(processedPath[s].getX() *
      10.0) / 10.0, round(processedPath[s].getY() * 10.0) / 10.0,
00070             round(processedPath[s].getZ() * 10.0) / 10.0, round(processedPath[s].getA() * 10.0) /
      10.0, round(processedPath[s].getB() * 10.0) / 10.0,
00071             round(processedPath[s].getC() * 10.0) / 10.0);
00072     }
00073
00074     fputs("END", fid);
00075 }
```

Benutzt postProcessing(), processedPath, speed und speedManual.

Wird benutzt von main().

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 77 der Datei RobCodeGenerator.cpp.

```
00078 {
00079     COutputPoint3D p;
00080     double timePrev = 1;
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 ueberschritten 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, path[s]);
00107         timePrev = path[s].getTime();
00108         processedPath.push_back(p);
00109     }
00110 
00111 }
```

Benutzt A, B, C, calculateAngles(), calculateSpeed(), MAX_SPEED, orientationManual, processedPath, CPoint3D::set(), COutputPoint3D::setA(), COutputPoint3D::setB(), COutputPoint3D::setC(), COutputPoint3D::setSpeed(), speed und speedManual.

Wird benutzt von generateRobCode().

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.

Wird benutzt von CRobCodeGenerator(), CRobCodeGenerator() und postProcessing().

5.11.4.2 B

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

User eingegebener Winkel B

Definiert in Zeile 102 der Datei RobCodeGenerator.h.

Wird benutzt von CRobCodeGenerator(), CRobCodeGenerator() und postProcessing().

5.11.4.3 C

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

User eingegebener Winkel C

Definiert in Zeile 106 der Datei RobCodeGenerator.h.

Wird benutzt von CRobCodeGenerator(), CRobCodeGenerator() und postProcessing().

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.

Wird benutzt von CRobCodeGenerator(), CRobCodeGenerator() und postProcessing().

5.11.4.5 processedPath

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

Fertig bearbeiteter Pfad

Definiert in Zeile 82 der Datei RobCodeGenerator.h.

Wird benutzt von calculateSpeed(), generateRobCode() und postProcessing().

5.11.4.6 speed

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

User eingegebene Geschwindigkeit

Definiert in Zeile 86 der Datei RobCodeGenerator.h.

Wird benutzt von calculateSpeed(), CRobCodeGenerator(), CRobCodeGenerator(), generateRobCode() und postProcessing().

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.

Wird benutzt von CRobCodeGenerator(), CRobCodeGenerator(), generateRobCode() und postProcessing().

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 22 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     /* Douglas Peucker für Segmente aufrufen*/
00026     for (size_t s = 0; s < segments.size(); s++)
00027     {
00028         douglasPeuckerRecursive(segmentsApprox[s], segmentsApprox[s].begin(),
      --(segmentsApprox[s].end()));
00029     }
00030
00031     /*  Logging der Daten*/
00032     log.setStep(3);
00033     log.logData(segmentsApprox);
00034 }
```

Benutzt douglasPeuckerRecursive(), CLogging::logData(), segmentsApprox und CLogging::setStep().

Wird benutzt von main().

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

Definiert in Zeile 52 der Datei SegmentApproximator.cpp.

```
00053 {
00054     if (segment.size() < 3) return;  // min Größe pro Seg 3
00055     if (distance(startItr, endItr) == 2) return;        // Zeigerabstand == 2
00056     CInputPoint3D pStart; CInputPoint3D pEnd;     // Variablen deklarieren
00057
00058
00059     /* Startpunkt setzen */
00060     pStart.setX(startItr->getX()); pStart.setY(startItr->getY()); pStart.setZ(startItr->getZ());
00061     pStart.setTime(startItr->getTime());
00062     pStart.setEulerMatrix(startItr->getEulerMatrix());
00063
00064     /* Endpunkt setzen */
00065     pEnd.setX(endItr->getX()); pEnd.setY(endItr->getY()); pEnd.setZ(endItr->getZ());
00066     pEnd.setTime(endItr->getTime());
00067     pEnd.setEulerMatrix(endItr->getEulerMatrix());
00068
00069     double dist = 0.0, maxDist = 0.0;                   // dist und maxDist initialisieren
00070     std::list<CInputPoint3D>::iterator maxItr, itr;     // Zeiger bilden
00071
00072
00073     /* am weitesten Entfernten Punkt suchen */
00074     for (itr = startItr; itr != endItr; itr++)
00075     {
00076         CLine3D line = CLine3D(pStart, pEnd);
00077         // calc distance
00078         dist = itr->distanceTo(line);
00079         if (dist > maxDist) {
00080             maxDist = dist;
00081             maxItr = itr;
00082         }
```

```
00083     }
00084
00085     if (maxDist <= maxDistance) {
00086
00087         segment.erase((++startItr), endItr);        // Punkt löschen
00088         return;
00089     }
00090
00091     /*  Douglas Peucker erneut aufrufen */
00092     douglasPeuckerRecursive(segment, startItr, maxItr);
00093     douglasPeuckerRecursive(segment, maxItr, endItr);
00094 }
```

Benutzt douglasPeuckerRecursive(), maxDistance, CInputPoint3D::setEulerMatrix(), CInputPoint3D::setTime(), CPoint3D::setX(), CPoint3D::setY() und CPoint3D::setZ().

Wird benutzt von approx() und douglasPeuckerRecursive().

5.12.3.3 getmaxDistance()

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

Gibt die maximale Abweichung zurück.

Rückgabe

maxDistanceSource double

Definiert in Zeile 41 der Datei SegmentApproximator.cpp.

```
00042 {
00043     return maxDistance;      // Rueckgabe von maxDistance
00044 }
```

Benutzt maxDistance.

5.12.3.4 getSegmentsApproxVector()

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

Gibt den bereinigten Pfad zurück.

Rückgabe

vector<list<CInputPoint3D>>&

Definiert in Zeile 46 der Datei SegmentApproximator.cpp.

```
00047 {
00048     return segmentsApprox;        // Rueckgabe der Segmente
00049 }
```

Benutzt segmentsApprox.

Wird benutzt von main().

5.12.3.5 setmaxDistance()

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

Setzt die maximale Abweichung.

Parameter

maxDistanceSource	double

Definiert in Zeile 36 der Datei SegmentApproximator.cpp.

```
00037 {
00038     maxDistance = maxDistanceSource;     // setze maxDistance
00039 }
```

Benutzt maxDistance.

Wird benutzt von main().

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 67 der Datei SegmentApproximator.h.

Wird benutzt von douglasPeuckerRecursive(), getmaxDistance() und setmaxDistance().

5.12.4.2 segmentsApprox

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

Bereinigten Pfad

Definiert in Zeile 63 der Datei SegmentApproximator.h.

Wird benutzt von approx() und getSegmentsApproxVector().

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.

Makrodefinitionen

- #define _USE_MATH_DEFINES

6.1.1 Ausführliche Beschreibung

Header File handling Euler Matrix.

Definiert in Datei EulerMatrix.h.

6.1.2 Makro-Dokumentation

6.1.2.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

Definiert in Zeile 10 der Datei EulerMatrix.h.

6.2 EulerMatrix.h

gehe zur Dokumentation dieser Datei

```
00001
00007 #include <tuple>
00008 #include <cmath>
00009
00010 #define _USE_MATH_DEFINES
00011
00012 using namespace std;
00013
00014 #pragma once
00015
00020 class CEulerMatrix
00021 {
00022 public:
00028     CEulerMatrix(void);
00035     CEulerMatrix(float inputMatrix[3][3]);
00039     ~CEulerMatrix();
00040
00045     void setMatrix(float inputMatrix[3][3]);
00050     CEulerMatrix getEulerMatrix(void);
00051
00056     void getMatrix(float Matrix[][3]);
00064     CEulerMatrix angels2mat(double A, double B, double C);
00065
00070     tuple<double , double , double> calculateAngels(void);
00071
00072 private:
00076     float eulerMatrix[3][3];
00077
00078 };
00079
```

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 filepath, 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 #include <vector>
00008 #include <list>
00009 #include <iostream>
00010 #include <math.h>
00011 #include "Point3D.h"
00012 #include "Logging.h"
00013
00014 using namespace std;
00015
00016 #pragma once
00017
00022 class CSegmentApproximator
00023 {
00024 public:
00029     CSegmentApproximator(void);
00033     ~CSegmentApproximator(void);
00034
00041     void approx(const vector<list<CInputPoint3D»& Segments, CLogging log);
00046     void setmaxDistance(double maxDistanceSource);
00051     double getmaxDistance();
00052
00057     vector<list<CInputPoint3D»& getSegmentsApproxVector();
00058
00059 private:
00063     vector<list<CInputPoint3D» segmentsApprox;
00067     double maxDistance;
00068
00077     void douglasPeuckerRecursive(list<CInputPoint3D>& segment, std::list<CInputPoint3D>::iterator
      startItr, std::list<CInputPoint3D>::iterator endItr);
00078 };
```

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;         // EulerMatrix zurÃ¼ck geben
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];     // DummyMatrix erstellen
00068
00069     /* Berechnung der Matrix */
00070
00071     Matrix[0][0] = cos(A) * cos(C) - sin(A) * cos(B) * sin(C);
00072     Matrix[0][1] = -cos(A) * sin(C) - sin(A) * cos(B) * cos(C);
00073     Matrix[0][2] = sin(A) * sin(B);
00074
00075     Matrix[1][0] = sin(A) * cos(C) + cos(A) * cos(B) * sin(C);
00076     Matrix[1][1] = -sin(A) * sin(C) + cos(A) * cos(B) * cos(C);
00077     Matrix[1][2] = -cos(A) * sin(B);
00078
00079     Matrix[2][0] = sin(B) * sin(C);
00080     Matrix[2][1] = sin(B) * cos(C);
00081     Matrix[2][2] = cos(B);
00082
00083     CEulerMatrix buffer(Matrix);        // DummyMatrix in DummyEulerMatrix schreiben
00084     return buffer;                      // Matrix zurÃ¼ck geben
00085 }
00086
00087 //TODO: Kommentar
00088 tuple<double , double , double> CEulerMatrix::calculateAngels(void)
00089 {
00090     double a, b, c, sin_a, cos_a, sin_b, abs_cos_b, sin_c, cos_c;
00091
00092     /*
00093     a == Winkel Alpha
00094     b == Winkel Beta
00095     c == Winkel Gamma
00096
00097     sin_a == sinus alpha
00098     cos_a == cosinus alpha
00099     sin_b == Matrix[2][0] * -1
00100     abs_cos_b == ??
00101     sin_c == sinus gamma
00102     cos_c == cosinus gamma
00103     */
00104
00105
00106     /* Berechnung von alpha*/
00107     a = atan2(eulerMatrix[1][0], eulerMatrix[0][0]);
00108
00109     /* Berechung von beta */
00110     sin_a = sin(a);
00111     cos_a = cos(a);
00112     sin_b = eulerMatrix[2][0] * -1;
00113     abs_cos_b = cos(a) * eulerMatrix[0][0] + sin(a) * eulerMatrix[1][0];
00114
00115     b = atan2 (sin_b, abs_cos_b);
00116
00117     /* Berechung von gamma */
00118     sin_c =  sin_a * eulerMatrix[0][2] - cos_a * eulerMatrix[1][2];
00119     cos_c = -sin_a * eulerMatrix[0][1] + cos_a * eulerMatrix[1][1];
00120
00121     c = atan2(sin_c, cos_c);
00122
00123     /* Bogenmass in Gradmass umrechnen */
00124     a = a * 180 / M_PI;
00125     b = b * 180 / M_PI;
00126     c = c * 180 / M_PI;
00127
00128
00129     return make_tuple(a, b, c);     // RÃ¼ckgabe der Winkel
00130 }
00131
```

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
00011 /* CInputParamameter mir Übergabewerten initialisieren */
00012 CInputParameter::CInputParameter(double initSpeed, bool initSpeedManual, bool initOrientationManual,
      double initA, double initB, double initC)
00013 {
00014     speed = initSpeed;
00015     speedManual = initSpeedManual;
00016     orientationManual = initOrientationManual;
00017     A = initA;
00018     B = initB;
00019     C = initC;
00020
00021 }
00022
00023 /* CInputParameter mit 0 initialisieren */
00024 CInputParameter::CInputParameter(void)
00025 {
00026     speed = 0.1;
00027     A = 0;
00028     B = 75;
00029     C = 0;
00030     speedManual = true,
00031     orientationManual = true;
00032
00033 }
00034
00035 CInputParameter::~CInputParameter(void)
00036 {
00037
00038 }
00039
00040
```

```
00041 /* Einstellung für Orientierung und Winkel setzten */
00042 void CInputParameter::setOrientation(bool initOrientationManual, double initA, double initB, double
      initC)
00043 {
00044     orientationManual = initOrientationManual;
00045     A = initA;
00046     B = initB;
00047     C = initC;
00048 }
00049
00050 /* Einstellung für Geschwindigkeit und Geschwindigkeit setzen */
00051 void CInputParameter::setSpeed(double initSpeed, bool initSpeedManual)
00052 {
00053     speed = initSpeed;
00054     speedManual = initSpeedManual;
00055 }
00056
00057 vector<list<CInputPoint3D»& CInputParameter::getPath()
00058 {
00059     return initialPath;      // Path zurück geben
00060 }
00061
00062 double CInputParameter::getSpeed(void)
00063 {
00064     return speed;          // Geschwindigkeit zurück geben
00065 }
00066
00067 bool CInputParameter::getSpeedManual(void)
00068 {
00069     return speedManual;              // Vorgewählte Einstellung für Geschwindigkeit zurück geben
00070 }
00071
00072 bool CInputParameter::getOrientationManual(void)
00073 {
00074     return orientationManual;      // Vorgewählte Einstellung für Orientierung zurück geben
00075 }
00076
00077 tuple <double, double, double> CInputParameter::getAngles(void)
00078 {
00079     return make_tuple(A, B, C);      // Winkel zurück geben
00080 }
00081
00082
00083 /* Eingabedatei öffnen */
00084 void CInputParameter::openFile(string path)
00085 {
00086     ifstream fin(path);
00087     CInputPoint3D tmpPoint;        // Zwischenspeicher zum konvertieren von tmpEuler in Point3D
00088     CEulerMatrix tmpEuler;      // Zwischenspeicher zum konverteiren von DummyMatrix in EulerMatrix
00089     double x, y, z;             // Punktkoordinaten
00090     double x_prev = 0, y_prev = 0, z_prev = 0;      // Zwischenspeicher für Punktkoordinaten
00091     double timestamp;           // Zeitstempel
00092     int segmentCount = -1;      // Segmentzähler
00093     float dummyMatrix[3][3];    // DummyMatrix zum speichern
00094
00095
00096     if (!fin.is_open())
00097     {
00098         cerr « "Datei konnte nicht geöffnet werden" « endl;       // Fehler Datei konnte nicht
      geöffnet werden.
00099     }
00100     string line;
00101
00102     while(getline(fin, line))
00103     {
00104         std::istringstream sStream (line);
00105         sStream » timestamp » x » y » z » dummyMatrix[0][0] » dummyMatrix[0][1] » dummyMatrix[0][2]
      // Zeile in die einzelnen Parameter zerlegen
00106             » dummyMatrix[1][0] » dummyMatrix[1][1] » dummyMatrix[1][2] » dummyMatrix[2][0] »
      dummyMatrix[2][1] » dummyMatrix[2][2];      // und in DummyMatrix bzw. Variablen abspeichern
00107
00108         tmpEuler.setMatrix(dummyMatrix);                                    // DummyMatrix[3][3] in
      EulerMatrix speichern
00109         tmpPoint.setPoint(timestamp, x, y, z, tmpEuler.getEulerMatrix()); // Variablen und EulerWinkel
      in CPoint3D speichern
00110
00111         if (detectJump(tmpPoint, x_prev, y_prev, z_prev)) // if there is a jump in the data, start da
      new segment
00112         {
00113             segmentCount++;                                 // neues Segement anlegen
00114             initialPath.push_back(list<CInputPoint3D>());    // Punkt in Segent speichern
00115         }
00116
00117         initialPath[segmentCount].push_back(tmpPoint);       // Punkt in bestehendes Segment
      abspeichern
00118
00119         x_prev = x;     // X-Wert zwischenspeichern
```

```
00120         y_prev = y;     // Y-Wert zwischenspeichern
00121         z_prev = z;     // Z-Wert zwischenspeichern
00122     }
00123     fin.close();        // Datei schließen
00124 }
00125
00126 bool CInputParameter::detectJump(CInputPoint3D p, double x_prev, double  y_prev, double z_prev)
00127 {
00128     if(abs(p.getX() - x_prev) > difference)               // Abstand zwischen Punkten größer max
      Differenz??
00129         return true;
00130     else if(abs(p.getY() - y_prev) > difference)      // Abstand zwischen Punkten größer max
      Differenz??
00131         return true;
00132     else if(abs(p.getZ() - z_prev) > difference)          // Abstand zwischen Punkten größer max
      Differenz??
00133         return true;
00134     else
00135         return false;
00136 }
```

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 /* initialisieren mit 2 Punkten*/
00015 CLine3D::CLine3D(CPoint3D P1, CPoint3D P2)
00016 {
00017     p1 = P1;
00018     p2 = P2;
00019 }
00020
00021 CLine3D::~CLine3D(void)
00022 {
00023 }
```

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 /* Step mit 0 initialisiren */
00010 CLogging::CLogging(void)
00011 {
00012     step = 0;
00013 }
00014
00015 /* Path mit Parameter initialisieren*/
00016 CLogging::CLogging(string Path)
00017 {
00018     path = Path;
00019 }
00020
00021 CLogging::~CLogging(void)
00022 {
00023
00024 }
00025
00026 void CLogging::setStep(int Step)
00027 {
00028     step = Step;    // Step setzen
00029 }
00030
00031 void CLogging::logData(vector<list<CInputPoint3D»& sourcePath)
00032 {
00033     string filepath;            // file Pfad
00034     float dummyMatrix[3][3];    // dummyMatrix zum Zwischenspeichern
00035     CEulerMatrix tmpEuler;      // CEulerMatrix zum Zwischenspeichern
00036
00037     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
00038
00039     FILE* fid = fopen(filepath.c_str(), "w");       // file öffnen
00040
00041     if (fid == NULL)
00042     {
00043         cerr « "ERROR - Can NOT write to output file!\n";      // Fehler beim file öffnen
00044         return;
00045     }
00046
00047     for (size_t s = 0; s < sourcePath.size(); s++) //for all segments
00048     {
00049         list<CInputPoint3D>::iterator itr = sourcePath[s].begin();
00050
00051         tmpEuler = itr->getEulerMatrix();
00052         tmpEuler.getMatrix(dummyMatrix);
00053
00054         /* Ausgeben der Punkte mit dummyMatrix */
00055         for (; itr != sourcePath[s].end(); itr++) //for all points in the segment
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],
00059                 dummyMatrix[1][0], dummyMatrix[1][1], dummyMatrix[1][2],
00060                 dummyMatrix[2][0], dummyMatrix[2][1], dummyMatrix[2][2]);
00061         }
00062
00063         itr--;
00064     }
00065 }
00066
00067 void CLogging::logData(vector<CInputPoint3D>& sourcePath)
00068 {
00069     string filepath;            // file Pfad
00070     float dummyMatrix[3][3];    // dummyMatrix zum Zwischenspeichern
00071     CEulerMatrix tmpEuler;      // CEulerMatrix zum Zwischenspeichern
00072
00073     filepath = path + "/" + "0" + std::to_string(step) + "_path.csv";
```

```
00074
00075     FILE* fid = fopen(filepath.c_str(), "w");   // file öffnen
00076
00077     if (fid == NULL)
00078     {
00079         cerr « "ERROR - Can NOT write to output file!\n";  // Fehler beim file öffnen
00080         return;
00081     }
00082
00083     /* Ausgeben der Punkte mit dummyMatrix */
00084     for (size_t s = 0; s < sourcePath.size(); s++) //for all points in the vector
00085     {
00086         tmpEuler.getMatrix(dummyMatrix);
00087
00088         fprintf(fid, "%f %f %f %f %f %f %f %f %f %f %f %f\n", (double)sourcePath[s].getTime(),
00089             (double)sourcePath[s].getX(), (double)sourcePath[s].getY(), (double)sourcePath[s].getZ(),
00090             dummyMatrix[0][0], dummyMatrix[0][1], dummyMatrix[0][2],
00091             dummyMatrix[1][0], dummyMatrix[1][1], dummyMatrix[1][2],
00092             dummyMatrix[2][0], dummyMatrix[2][1], dummyMatrix[2][2]);
00093     }
00094 }
```

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;           // initialisieren mit Standardfenstergröße 3
00014 }
00015
00016 CMeanFilter::CMeanFilter(int Window)
00017 {
00018     windowSize = Window;  // initialisieren der Fenstergröße mit Übergabewert
00019 }
00020
00021 CMeanFilter::~CMeanFilter()
00022 {
00023 }
00024
00025 void CMeanFilter::setWindowSize(int Window)
00026 {
00027     windowSize = Window;  // setzen der Fenstergröße mit Übergabewert
00028 }
00029
00030 int CMeanFilter::getWindowSize()
00031 {
00032     return windowSize;        // Fenstergröße zurück geben
```

```
00033 }
00034
00035 vector<list<CInputPoint3D»& CMeanFilter::getPath()
00036 {
00037     return meanPath;        // Mittelwert zurück geben
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;     // Variablen zum Speichern der Summe
00055     double div = 0;                          // Variable zum Speichern des Teilers
00056
00057     CInputPoint3D p;       //Point3D zum Zwischenspeichern
00058
00059     size_t inputSize = segment.size();
00060
00061     list<CInputPoint3D>::iterator it = segment.begin();
00062     list<CInputPoint3D> newSegment;
00063
00064     for (size_t i = 0; i < inputSize - windowSize; ++i) //For each element in the Segment
00065     {
00066         sumX = 0, sumY = 0, sumZ = 0;    // Variablen zum Speichern der Summe auf 0 zurück setzen
00067         div = 0;                         // Variable zum Speichern des Teilers auf 0 zurück setzen
00068         p.setTime(it->getTime());
00069         p.setEulerMatrix(it->getEulerMatrix());
00070         for (size_t j = i; j < i + windowSize; ++j) // Build the sums for the three points
00071         {
00072             sumX += it->getX();
00073             sumY += it->getY();
00074             sumZ += it->getZ();
00075             div++;
00076             it++;
00077         }
00078         for (size_t index = windowSize; index > 0; index--) // Pain, the iterator has to be set back
00079         {
00080             it--;
00081         }
00082         p.set(sumX / div, sumY / div, sumZ / div); // Calculate smoothed values
00083         if(it != segment.end())
00084             it++;
00085         newSegment.push_back(p);
00086     }
00087     return newSegment;
00088 }
```

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
00027     for (size_t s = 0; s < segments.size(); s++) //for all segments
00028     {
00029         list<CInputPoint3D>::iterator itr = segments[s].begin();
00030
00031         for (; itr != segments[s].end(); itr++) //for all points in the segment
00032         {
00033             point.set((double)itr->getX(), (double)itr->getY(), (double)itr->getZ());
00034             point.setTime(itr->getTime());
00035             point.setEulerMatrix(itr->getEulerMatrix());
00036             path.push_back(point);
00037         }
00038
00039         itr--;
00040     }
00041
00042     log.setStep(4);
00043     log.logData(path);
00044 }
```

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 mit 0-Werten */
00013 CPoint3D::CPoint3D(void)
00014 {
00015     x = 0;
00016     y = 0;
00017     z = 0;
00018 }
00019
00020 /* initialiseren des Punktes mit Koordinaten */
00021 CPoint3D::CPoint3D(double X, double Y, double Z)
00022 {
00023     x = X;
00024     y = Y;
00025     z = Z;
00026 }
00027
00028 CPoint3D::~CPoint3D(void)
00029 {
00030 }
00031
00032 double CPoint3D::getX(void)
00033 {
00034     return x;  // X-Koordinate zurück geben
00035 }
00036
00037 double CPoint3D::getY(void)
00038 {
00039     return y;  // Y-Koordinate zurück geben
00040 }
00041
00042 double CPoint3D::getZ(void)
00043 {
00044     return z;  // Z-Koordinate zurück geben
00045 }
00046
00047 void CPoint3D::setX(double X)
00048 {
00049     x = X; // X-Koordinate setzen
00050 }
00051
00052 void CPoint3D::setY(double Y)
00053 {
00054     y = Y; // Y-Koordinate setzen
00055 }
00056
00057 void CPoint3D::setZ(double Z)
00058 {
00059     z = Z; // Z-Koordinate setzen
00060 }
00061
00062 /* X-, Y- und Z-Koordinate setzen */
00063 void CPoint3D::set(double X, double Y, double Z)
00064 {
00065     x = X; // X-Koordinate setzen
00066     y = Y; // Y-Koordinate setzen
00067     z = Z; // Z-Koordinate setzen
00068 }
00069
00070 /* Distanz zwischen Punkt und übergebenen Punkt berechnen */
00071 double CPoint3D::distanceTo(CPoint3D point)
00072 {
00073     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
00074 }
00075
00076 double CPoint3D::distanceTo(CLine3D line)
00077 {
00078     double bx, by, bz, rv_sq, dist, vp1, vp2, vp3;          // Variablen Anlegen
00079
00080     /*
00081     Vermessen wird der Punkt selbst
00082
00083     bx, by, bz      == Vektordifferenz
00084     rv_sq           == Betrag des Linienvektors
00085     dist            == Distanz von Punkt zu Linie
00086     vp1, vp2, vp3   == Vektorprodukte
```

```
00087     */
00088
00089     int rvx = line.p1.x - line.p2.x;      // Parameter X des Linienvektor berechnen
00090     int rvy = line.p1.y - line.p2.y;      // Parameter Y des Linienvektor berechnen
00091     int rvz = line.p1.z - line.p2.z;      // Parameter Z des Linienvektor berechnen
00092
00093     rv_sq = sqrt(((double)rvx * (double)rvx) + ((double)rvy * (double)rvy) + ((double)rvz *
      (double)rvz));      // Betrag des Linienvektor berechnen
00094
00095     bx = x - (double)line.p1.x;             // X(Punkt) - X(Aufpunkt)
00096     by = y - (double)line.p1.y;             // Y(Punkt) - Y(Aufpunkt)
00097     bz = z - (double)line.p1.z;             // Z(Punkt) - Z(Aufpunkt)
00098
00099     vp1 = by * rvz - bz * rvy;              // Parameter X Vektorprodukt
00100     vp2 = bz * rvx - bx * rvz;              // Parameter Y Vektorprodukt
00101     vp3 = bx * rvy - by * rvx;              // Parameter Z Vektorprodukt
00102
00103     dist = sqrt(vp1 * vp1 + vp2 * vp2 + vp3 * vp3) / rv_sq; // Betrag des Vektors berechnen
00104
00105     return dist;
00106 }
00107
00108 // InputPoint3D
00109
00110 CInputPoint3D::CInputPoint3D(void) : CPoint3D()
00111 {
00112     timestamp = 0;      // Zeitstempel mit 0 initialisieren
00113 }
00114
00115 /* Initialisieren des Punktes mit Parameter */
00116 CInputPoint3D::CInputPoint3D(double X, double Y, double Z, double Timestamp, CEulerMatrix Matrix)
00117 {
00118     x = X;
00119     y = Y;
00120     z = Z;
00121     timestamp = Timestamp;
00122     orientationMatrix = Matrix;
00123 }
00124
00125 CInputPoint3D::~CInputPoint3D(void)
00126 {
00127 }
00128
00129 void CInputPoint3D::setEulerMatrix(CEulerMatrix orientation)
00130 {
00131     orientationMatrix = orientation;    // EulerMatrix setzen
00132 }
00133
00134
00135 void CInputPoint3D::setPoint(double time, double X, double Y, double Z, CEulerMatrix orientation)
00136 {
00137     setTime(time);    // Zeitstempel setzen
00138     set(X, Y, Z);     // setze Punkt-Koordinaten
00139     setEulerMatrix(orientation); // EulerMatrix setzen
00140 }
00141
00142 void CInputPoint3D::setTime(double time)
00143 {
00144     timestamp = time;  // Zeitstempel setzen
00145 }
00146
00147 CEulerMatrix CInputPoint3D::getEulerMatrix()
00148 {
00149     return orientationMatrix;  // Rückgabe der EulerMatrix
00150 }
00151
00152 double CInputPoint3D::getTime()
00153 {
00154     return timestamp;       // Rückgabe des Zeitstempel
00155 }
00156
00157 // OutputPoint3D
00158 /* Punkt mit 0 initialisieren */
00159 COutputPoint3D::COutputPoint3D(void) : CPoint3D()
00160 {
00161     speed = 0;
00162     a = 0;
00163     b = 0;
00164     c = 0;
00165 }
00166
00167 /* Punkt mit Parameter initialisieren*/
00168 COutputPoint3D::COutputPoint3D(double Speed, double X, double Y, double Z, double A, double B, double
      C)
00169 {
00170     speed = Speed;
00171     a = A;
```

```
00172     b = B;
00173     c = C;
00174     x = X;
00175     y = Y;
00176     z = Z;
00177 }
00178
00179 COutputPoint3D::~COutputPoint3D(void)
00180 {
00181
00182 }
00183
00184 double COutputPoint3D::getA(void)
00185 {
00186     return a;  // Rückgabe Winkel alpha
00187 }
00188
00189 double COutputPoint3D::getB(void)
00190 {
00191     return b;  // Rückgabe Winkel beta
00192 }
00193
00194 double COutputPoint3D::getC(void)
00195 {
00196     return c;  // Rückgabe Winkel gamma
00197 }
00198
00199 double COutputPoint3D::getSpeed(void)
00200 {
00201     return speed;  // Rückgabe Geschwindigkeit
00202 }
00203
00204 void COutputPoint3D::setA(double A)
00205 {
00206     a = A;     // setze Winkel alpha
00207 }
00208
00209 void COutputPoint3D::setB(double B)
00210 {
00211     b = B;     // setze Winkel beta
00212 }
00213
00214 void COutputPoint3D::setC(double C)
00215 {
00216     c = C;     // setze Winkel gamma
00217 }
00218
00219 void COutputPoint3D::setSpeed(double Speed)
00220 {
00221     speed = Speed;     // setze Geschwindigkeit
00222 }
```

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 mit 0 initialiseren */
00011 CRobCodeGenerator::CRobCodeGenerator(void)
00012 {
00013     speed = 0;
00014     speedManual = 0;
00015     orientationManual = 0;
00016     A = 0;
00017     B = 0;
00018     C = 0;
00019 }
00020
00021 /* CRobCodeGenerator mit Uebergabewerten initialisieren */
00022 CRobCodeGenerator::CRobCodeGenerator(double Speed, bool SpeedManual, bool OrientationManual,
      tuple<double, double, double> angles)
00023 {
00024     speed = Speed;
00025     speedManual = SpeedManual;
00026     orientationManual = OrientationManual;
00027     A = get<0>(angles);
00028     B = get<1>(angles);
00029     C = get<2>(angles);
00030 }
00031
00032 CRobCodeGenerator::~CRobCodeGenerator(void)
00033 {
00034 }
00035
00036 void CRobCodeGenerator::generateRobCode(vector<CInputPoint3D>& points, string filepath, string
      filename)
00037 {
00038     postProcessing(points); // Calculates all the necessary values
00039
00040     errno_t err;
00041
00042     FILE* fid;
00043
00044     string fullPath = filepath + "/" + filename;
00045
00046     if ((err = fopen_s(&fid, fullPath.c_str(), "w")) != 0) // Errorhandling for File opening
00047     {
00048         string msg = "Open file: ";
00049         msg += filename;
00050         msg += " failed!";
00051
00052         throw exception(msg.c_str());
00053     }
00054
00055     filename.erase(filename.end()-4,filename.end());        // lÃ¶scht .src
00056     fprintf(fid, "DEF %s \n", filename.c_str());            // DEF in file schreiben
00057
00058     fputs("PTP $POS_ACT\n", fid);                           // PTP zur aktuellen Position in file
      schreiben
00059
00060     if (speedManual) // If the speed is set to manual, it will be defined once at the beginning of the
      file
00061     {
00062         fprintf(fid, "$VEL.CP = %f\n", speed);     // Geschwindigkeit ein file schreiben
00063     }
00064
00065     for (size_t s = 0; s < points.size(); s++)
00066     {
00067         if (!speedManual) // If the speed is calculated it needs to be before every LIN command
00068             fprintf(fid, "&VEL.CP = %f\n", (float)processedPath[s].getSpeed());
00069         fprintf(fid, "LIN {X %f, Y %f, Z %f, A %f, B %f, C %f}\n", round(processedPath[s].getX() *
      10.0) / 10.0, round(processedPath[s].getY() * 10.0) / 10.0,
00070             round(processedPath[s].getZ() * 10.0) / 10.0, round(processedPath[s].getA() * 10.0) /
      10.0, round(processedPath[s].getB() * 10.0) / 10.0,
00071             round(processedPath[s].getC() * 10.0) / 10.0);
00072     }
00073
00074     fputs("END", fid);
00075 }
00076
00077 void CRobCodeGenerator::postProcessing(vector<CInputPoint3D>& path)
00078 {
00079     COutputPoint3D p;
00080     double timePrev = 1;
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 ueberschritten 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, path[s]);
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(1.5);
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 }
```

Benutzt CSegmentApproximator::approx(), CPathBuilder::createPath(), CRobCodeGenerator::generateRobCode(), CInputParameter::getAngles(), CInputParameter::getOrientationManual(), CInputParameter::getPath(), CMeanFilter::getPath(), CPathBuilder::getPath(), CSegmentApproximator::getSegmentsApproxVector(), CInputParameter::getSpeed(), CInputParameter::getSpeedManual(), CMeanFilter::mean(), CInputParameter::openFile(), CSegmentApproximator::setmaxDistance() und CMeanFilter::setWindowSize().

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(1.5);
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     /* Douglas Peucker für Segmente aufrufen*/
00026     for (size_t s = 0; s < segments.size(); s++)
00027     {
00028         douglasPeuckerRecursive(segmentsApprox[s], segmentsApprox[s].begin(),
    --(segmentsApprox[s].end()));
00029     }
00030
00031     /*  Logging der Daten*/
00032     log.setStep(3);
00033     log.logData(segmentsApprox);
00034 }
00035
00036 void CSegmentApproximator::setmaxDistance(double maxDistanceSource)
00037 {
00038     maxDistance = maxDistanceSource;     // setze maxDistance
00039 }
00040
00041 double CSegmentApproximator::getmaxDistance()
00042 {
00043     return maxDistance;      // Rueckgabe von maxDistance
00044 }
00045
00046 vector<list<CInputPoint3D>& CSegmentApproximator::getSegmentsApproxVector()
00047 {
00048     return segmentsApprox;        // Rueckgabe der Segmente
00049 }
00050
00051 //TODO: Kommentar
00052 void CSegmentApproximator::douglasPeuckerRecursive(list<CInputPoint3D>& segment,
    std::list<CInputPoint3D>::iterator startItr, std::list<CInputPoint3D>::iterator endItr)
00053 {
00054     if (segment.size() < 3) return;  // min Größe pro Seg 3
00055     if (distance(startItr, endItr) == 2) return;        // Zeigerabstand == 2
00056     CInputPoint3D pStart; CInputPoint3D pEnd;     // Variablen deklarieren
00057
00058
00059     /* Startpunkt setzen */
00060     pStart.setX(startItr->getX()); pStart.setY(startItr->getY()); pStart.setZ(startItr->getZ());
00061     pStart.setTime(startItr->getTime());
00062     pStart.setEulerMatrix(startItr->getEulerMatrix());
00063
00064     /* Endpunkt setzen */
00065     pEnd.setX(endItr->getX()); pEnd.setY(endItr->getY()); pEnd.setZ(endItr->getZ());
00066     pEnd.setTime(endItr->getTime());
00067     pEnd.setEulerMatrix(endItr->getEulerMatrix());
00068
00069     double dist = 0.0, maxDist = 0.0;                   // dist und maxDist initialisieren
00070     std::list<CInputPoint3D>::iterator maxItr, itr;     // Zeiger bilden
00071
00072
```

```
00073     /* am weitesten Entfernten Punkt suchen */
00074     for (itr = startItr; itr != endItr; itr++)
00075     {
00076         CLine3D line = CLine3D(pStart, pEnd);
00077         // calc distance
00078         dist = itr->distanceTo(line);
00079         if (dist > maxDist) {
00080             maxDist = dist;
00081             maxItr = itr;
00082         }
00083     }
00084
00085     if (maxDist <= maxDistance) {
00086
00087         segment.erase((++startItr), endItr);        // Punkt löschen
00088         return;
00089     }
00090
00091     /*  Douglas Peucker erneut aufrufen */
00092     douglasPeuckerRecursive(segment, startItr, maxItr);
00093     douglasPeuckerRecursive(segment, maxItr, endItr);
00094 }
```

Index