PathTrackingAndInductionOfTheRobot

構築: Doxygen 1.8.6

2015 年 12 月 19 日 (土) 20 時 25 分 30 秒

Contents

Chapter 1

クラス索引

1.1 クラス一覧

クラス・構造体・共用体・インターフェースの一覧です。

Chapter 2

ファイル索引

2.1 ファイル一覧

ファイル一覧です。

Chapter 3

クラス詳解

3.1 AttitudeAngle 構造体

`#include <PathTrackingAndInductionOfTheRobot.hpp>`

公開変数類

- double **yaw**
- double **roll**
- double **pitch**

3.1.1 詳解

PathTrackingAndInductionOfTheRobot.hpp の 38 行目に定義があります。

3.1.2 メンバ詳解

3.1.2.1 double AttitudeAngle::pitch

PathTrackingAndInductionOfTheRobot.hpp の 41 行目に定義があります。
参照元 LeastSquareMethod::calcYawRollPitch(), EV3Control::set6DoFEV3().

3.1.2.2 double AttitudeAngle::roll

PathTrackingAndInductionOfTheRobot.hpp の 40 行目に定義があります。
参照元 LeastSquareMethod::calcYawRollPitch(), EV3Control::set6DoFEV3().

3.1.2.3 double AttitudeAngle::yaw

PathTrackingAndInductionOfTheRobot.hpp の 39 行目に定義があります。
参照元 LeastSquareMethod::calcYawRollPitch(), EV3Control::set6DoFEV3().
この構造体詳解は次のファイルから抽出されました:

- **PathTrackingAndInductionOfTheRobot.hpp**

3.2 DoF 構造体

`#include <PathTrackingAndInductionOfTheRobot.hpp>`

公開変数類

- double **x**
- double **y**
- double **z**
- double **yaw**
- double **roll**
- double **pitch**

3.2.1 詳解

PathTrackingAndInductionOfTheRobot.hpp の 29 行目に定義があります。

3.2.2 メンバ詳解

3.2.2.1 double DoF::pitch

PathTrackingAndInductionOfTheRobot.hpp の 35 行目に定義があります。
参照元 System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

3.2.2.2 double DoF::roll

PathTrackingAndInductionOfTheRobot.hpp の 34 行目に定義があります。
参照元 System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

3.2.2.3 double DoF::x

PathTrackingAndInductionOfTheRobot.hpp の 30 行目に定義があります。
参照元 System::saveDataContinuously(), System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

3.2.2.4 double DoF::y

PathTrackingAndInductionOfTheRobot.hpp の 31 行目に定義があります。
参照元 System::saveDataContinuously(), System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

3.2.2.5 double DoF::yaw

PathTrackingAndInductionOfTheRobot.hpp の 33 行目に定義があります。
参照元 System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

3.2.2.6 double DoF::z

PathTrackingAndInductionOfTheRobot.hpp の 32 行目に定義があります。

参照元 System::saveDataContinuously(), System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3().

この構造体詳解は次のファイルから抽出されました:

- **PathTrackingAndInductionOfTheRobot.hpp**

3.3 Drawing クラス

経路描画用のクラス

`#include <Drawing.hpp>`

公開メンバ関数

- **Drawing** ()
  コンストラクタ
- ~**Drawing** ()
  デストラクタ
- void **plot3D** (const string *outputDataName)
  3D 座標ファイルをプロットするメソッド
- void **gnuplotScript** (const string *dataFileName)
  メソッド*Drawing::gnuplotScript()*. 後で，*3D* 座標をプロットして確認するために *gnuplot* のスクリプトを出力する
- void **plot3DRealTime** (int countDataNum, **outputData outputData**[**OUTPUTDATA_MAX**])
- void **gnuplotScriptCoG** (const string *cogFileName)
  球の重心座標をプロットするメソッド *(c52)*
- void **gnuplotScriptEV3Unit** (Eigen::Vector3f coefficient_plane)
  EV3 の点群をプロットするためのスクリプト *(c78)*
- void **gnuplotScriptEV3Route** ()

3.3.1 詳解

経路描画用のクラス

Drawing.hpp の 20 行目に定義があります。

3.3.2 構築子と解体子

3.3.2.1 Drawing::Drawing ( )

コンストラクタ

メソッドDrawing::Drawing(). コンストラクタ

Drawing.cpp の 16 行目に定義があります。

```
17 {
18     if ((gnuplot = _popen("gnuplot", "w")) == NULL){
19         cout << "gnuplot が開けません. \"gnuplot/binary\"へパスが通っているか確認して下さい" << endl; //
    gnuplot/binary/gnuplot.exe を開く. ※ wgnuplot.exe は起動するが処理が進まず,gnuplot が起動したままになる
20     }
21
22     //(c43)
```

```
23     if ((gpr = _popen("gnuplot", "w")) == NULL){
24         cout << "gnuplot が開けません. \"gnuplot/binary\"へパスが通っているか確認して下さい" << endl; //
       gnuplot/binary/gnuplot.exe を開く. ※ wgnuplot.exe は起動するが処理が進まず,gnuplot が起動したままになる
25     }
26
27     //save_count = 0;
28 }
```

3.3.2.2 Drawing::∼Drawing ( )

デストラクタ

メソッドDrawing::Drawing(). デストラクタ

Drawing.cpp の 33 行目に定義があります。

```
34 {
35     _pclose(gnuplot);
36
37     //(c43)
38     //fprintf_s(gpr, "unset multiplot\n");
39     fprintf_s(gpr, "quit\n");
40     _pclose(gpr);
41 }
```

3.3.3 関数詳解

3.3.3.1 void Drawing::gnuplotScript (const string * *dataFileName*)

メソッドDrawing::gnuplotScript(). 後で，3D 座標をプロットして確認するために gnuplot のスクリプトを出力する

引数

dataFileName	

<gnuplot スクリプト用のポインタ

< ファイルのパスを格納する変数

Drawing.cpp の 105 行目に定義があります。

参照先 directoryName, NOC.

```
106 {
107     //if (checkNum == 1){ //データを保存する場合にスクリプトを生成する (c32)
108         FILE *gp;
109         char filePath[NOC];
110         char* scriptFileName = "splot.plt"; //出力するスクリプトファイルの名前 (c39)
111
112         sprintf_s(filePath, "%s/%s", directoryName, scriptFileName); //出力パスを固定
113         fopen_s(&gp, filePath, "w"); //ファイルを書き込みモードで開く
114
115         //共通設定
116         fprintf_s(gp, "set multiplot layout 2,2\n"); //1 つのウインドウに複数のプロットを表示
117         fprintf_s(gp, "set grid xtics ytics ztics\n");
118
119         //範囲設定 (自動縮尺)
120         if (plotMode == 4){
121             fprintf_s(gp, "set autoscale\n"); //自動縮尺 (c47)
122         }
123
124         //Time-X
125         fprintf_s(gp, "set title \"Time-X\"\n");
126         fprintf_s(gp, "set xlabel \"Time(ms)\"\n");
127         fprintf_s(gp, "set ylabel \"X-axis(mm)\"\n");
128         if (plotMode == 1){ //(1) 中心固定用
129             fprintf_s(gp, "set yrange [-5:5]\n"); //中心固定用
130         }
131         else if (plotMode == 2){ //(2) 縦移動用
132             fprintf_s(gp, "set yrange [-15:15]\n"); //縦移動用
133         }
134         else if(plotMode == 3){ //(3) 横移動用
135             fprintf_s(gp, "set yrange [-60:60]\n"); //横移動用
136         }
```

```
137         fprintf_s(gp, "plot \"%s\" using 1:2 with linespoints\n", dataFileName);
138
139         //Time-Y
140         fprintf_s(gp, "set title \"Time-Y\"\n");
141         fprintf_s(gp, "set xlabel \"Time(ms)\"\n");
142         fprintf_s(gp, "set ylabel \"Y-axis(mm)\"\n");
143         if (plotMode == 1){ //(1) 中心固定用
144             fprintf_s(gp, "set yrange [-10:0]\n"); //中心固定用
145         }
146         else if (plotMode == 2){ //(2) 縦移動用
147             fprintf_s(gp, "set yrange [-15:15]\n"); //縦移動用
148         }
149         else if(plotMode == 3){ //(3) 横移動用
150             fprintf_s(gp, "set yrange [-15:15]\n"); //横移動用
151         }
152         fprintf_s(gp, "plot \"%s\" using 1:3 with linespoints\n", dataFileName);
153
154         //Time-Z
155         fprintf_s(gp, "set title \"Time-Z\"\n");
156         fprintf_s(gp, "set xlabel \"Time(ms)\"\n");
157         fprintf_s(gp, "set ylabel \"Z-axis(mm)\"\n");
158         if (plotMode == 1){ //(1) 中心固定用
159             fprintf_s(gp, "set yrange [70:150]\n"); //中心固定用
160         }
161         else if (plotMode == 2){ //(2) 縦移動用
162             fprintf_s(gp, "set yrange [150:250]\n"); //縦移動用
163         }
164         else if(plotMode == 3){ //(3) 横移動用
165             fprintf_s(gp, "set yrange [50:250]\n"); //横移動用
166         }
167         fprintf_s(gp, "plot \"%s\" using 1:4 with linespoints\n", dataFileName);
168
169         fprintf_s(gp, "set xlabel \"X-axis(mm)\"\n");
170         fprintf_s(gp, "set ylabel \"Y-axis(mm)\"\n");
171         fprintf_s(gp, "set zlabel \"Z-axis(mm)\"\n");
172         fprintf_s(gp, "set title \"Path\"\n");
173         if (plotMode == 1){ //(1) 中心固定用
174             fprintf_s(gp, "set xrange [-10:10]\n");
175             fprintf_s(gp, "set yrange [-10:0]\n");
176             fprintf_s(gp, "set zrange [80:120]\n");
177         }
178         else if (plotMode == 2){ //(2) 縦移動用
179             fprintf_s(gp, "set xrange [-10:10]\n");
180             fprintf_s(gp, "set yrange [-10:10]\n");
181             fprintf_s(gp, "set zrange [50:300]\n");
182         }
183         else if(plotMode == 3){ //(3) 横移動用
184             fprintf_s(gp, "set xrange [-60:60]\n");
185             fprintf_s(gp, "set yrange [-10:10]\n");
186             fprintf_s(gp, "set zrange [150:250]\n");
187         }
188         fprintf_s(gp, "set view equal xy\n");
189         //fprintf_s(gp, "splot \"%s\" every 4 using 2:3:4 with linespoints\n", dataFileName);
    //点を間引いてプロットするとき (c40)
190         fprintf_s(gp, "splot \"%s\" using 2:3:4 with linespoints\n", dataFileName); //データをそのままプロット
(c40)
191
192         fprintf_s(gp, "unset multiplot\n");
193
194         fclose(gp);
195     //}
196     return;
197 }
```

3.3.3.2 void Drawing::gnuplotScriptCoG (const string * *cogFileName*)

球の重心座標をプロットするメソッド (c52)

メソッドDrawing::gnuplotScript(). 後で，球の重心座標をプロットするためにgnuplot のスクリプトを出力する

引数

dataFileName	

<gnuplot スクリプト用のポインタ

< ファイルのパスを格納する変数

Drawing.cpp の 249 行目に定義があります。

参照先 directoryName, NOC.

```
250 {
251     //if (checkNum == 1){ //データを保存する場合にスクリプトを生成する (c32)
252         FILE *gp;
253         char filePath[NOC];
254         char* scriptFileName = "cog.plt"; //出力するスクリプトファイルの名前 (c39)
255 
256         sprintf_s(filePath, "%s/%s", directoryName, scriptFileName); //出力パスを固定
257         fopen_s(&gp, filePath, "w"); //ファイルを書き込みモードで開く
258 
259         fprintf_s(gp, "set xlabel \"X-axis\"\n");
260         fprintf_s(gp, "set ylabel \"Y-axis\"\n");
261         fprintf_s(gp, "set zlabel \"Z-axis\"\n");
262         fprintf_s(gp, "set title \"Path\"\n");
263         if (plotMode == 1){ //(1) 中心固定用
264             fprintf_s(gp, "set xrange [-60:60]\n");
265             fprintf_s(gp, "set yrange [-30:30]\n");
266             fprintf_s(gp, "set zrange [50:300]\n");
267         }
268         else if (plotMode == 2){ //(2) 縦移動用
269             fprintf_s(gp, "set xrange [-10:10]\n");
270             fprintf_s(gp, "set yrange [-10:10]\n");
271             fprintf_s(gp, "set zrange [50:300]\n");
272         }
273         else if (plotMode == 3){ //(3) 横移動用
274             fprintf_s(gp, "set xrange [-60:60]\n");
275             fprintf_s(gp, "set yrange [-10:10]\n");
276             fprintf_s(gp, "set zrange [150:250]\n");
277         }
278         else{
279             fprintf_s(gp, "set autoscale\n"); //自動縮尺
280         }
281         fprintf_s(gp, "set view equal xy\n");
282         fprintf_s(gp, "splot \"%s\" using 2:3:4 with linespoints\n", cogFileName); //データをそのままプロット
(c40)
283         fclose(gp);
284     //}
285     return;
286 }
```

3.3.3.3 void Drawing::gnuplotScriptEV3Route ( )

Drawing.cpp の 316 行目に定義があります。

参照先 directoryName, NOC.

参照元 main().

```
317 {
318     FILE *fp;
319     char filepath_splot_ev3route[NOC];
320     sprintf_s(filepath_splot_ev3route, "data/%s/splot_ev3route.plt",
    directoryName);
321     fopen_s(&fp, filepath_splot_ev3route, "w");
322 
323     fprintf_s(fp, "set xlabel \"X-axis\"\n");
324     fprintf_s(fp, "set ylabel \"Y-axis\"\n");
325     fprintf_s(fp, "set zlabel \"Z-axis\"\n");
326     fprintf_s(fp, "set title \"EV3 Centroid Route\"\n");
327     fprintf_s(fp, "splot \"ev3route.dat\" with lp\n");
328 
329     fclose(fp);
330 
331     return;
332 }
```

3.3.3.4 void Drawing::gnuplotScriptEV3Unit (Eigen::Vector3f *coefficient_plane*)

EV3 の点群をプロットするためのスクリプト (c78)

メソッドDrawing::gnuplotScriptEV3Unit(). EV3 のユニットに関するデータファイルをプロットするスクリプトを生成するメソッド (c78)

Drawing.cpp の 291 行目に定義があります。

参照先 directoryName, NOC, save_count.

参照元 main().

```
292 {
293     FILE *fp; //ファイルストリームを開く
294     char filepath_splot_ev3[NOC];
295     sprintf_s(filepath_splot_ev3, "data/%s/%d/splot_ev3-%02d.plt", directoryName,
    save_count, save_count);
296     fopen_s(&fp, filepath_splot_ev3, "w"); //ファイルを開く
297
298     //(c78)
299     //if ((splot_ev3unit = _popen("gnuplot", "w")) == NULL){
300     //  cout << "gnuplot が開けません. \"gnuplot/binary\"へパスが通っているか確認して下さい" << endl;
     //gnuplot/binary/gnuplot.exe を開く. ※ wgnuplot.exe は起動するが処理が進まず,gnuplot が起動したままになる
301     //}
302
303     fprintf_s(fp, "set xlabel \"X-axis\"\n");
304     fprintf_s(fp, "set ylabel \"Y-axis\"\n");
305     fprintf_s(fp, "set zlabel \"Z-axis\"\n");
306     fprintf_s(fp, "set title \"PointCloud Plane(LSM) Centroid\"\n");
307     fprintf_s(fp, "splot \"dof6-%02d.dat\" pointsize 5,%f*x+%f*y+%f,\"point-%02d.dat\" every 5\n",
    save_count, coefficient_plane.x(),coefficient_plane.y(),coefficient_plane.z(),
    save_count);
308
309     //(c78)
310     //_pclose(splot_ev3unit);
311     fclose(fp);
312
313     return;
314 }
```

3.3.3.5 void Drawing::plot3D (const string * *outputDataName*)

3D 座標ファイルをプロットするメソッド

メソッドDrawing::plot3D().3D 座標をプロットするメソッド (未)

引数

outputDataName	

Drawing.cpp の 47 行目に定義があります。

参照先 directoryName.

```
48 {
49     //出力ファイル名の定義
50     char* plotImageName = "plot.jpeg"; //plot したときの画像ファイル名 (c39)
51     char* plotXYImageName = "plot_X-Y.jpeg"; //X-Y 平面でプロットした画像ファイルの名前 (c39)
52     char* plotXZImageName = "plot_X-Z.jpeg"; //X-Z 平面でプロットした画像ファイルの名前 (c39)
53     char* plotYZImageName = "plot_Y-Z.jpeg"; //Y-Z 平面でプロットした画像ファイルの名前 (c39)
54
55     //fprintf_s(gnuplot,"書式指定子", 変数); で書式を指定して, gnuplot へ出力できる
56     //fputs("コマンド",gnuplot); でコマンドをそのまま出力できる
57     fputs("set xlabel \"X-axis\"\n", gnuplot); //X 軸のラベル
58     fputs("set ylabel \"Y-axis\"\n", gnuplot); //Y 軸のラベル
59     fputs("set zlabel \"Z-axis\"\n", gnuplot); //Z 軸のラベル
60
61     fprintf_s(gnuplot, "splot \"%s/%s\" using 2:3:4 with lp\n", directoryName, outputDataName); //データをプ
ロット
62     fputs("set title \"Path\"\n", gnuplot); //グラフのタイトル
63     fputs("set term jpeg size 1280,720\n", gnuplot); //jpeg で保存するため
64     fprintf_s(gnuplot,"set output \"%s/%s\"\n", directoryName, plotImageName); //名前をつけて保存
65     fputs("rep\n", gnuplot); //画像の保存の反映
66
67     //fputs("set view 1,360,1,1\n", gnuplot); //(X,Y) 平面
68     fprintf_s(gnuplot, "plot \"%s/%s\" u 2:3 with lp\n", directoryName, outputDataName); //データをプロット
69     fputs("set title \"Path X-Y\"\n", gnuplot); //グラフのタイトル
70     fputs("set xlabel \"X-axis\"\n", gnuplot); //グラフの X 軸ラベル
71     fputs("set ylabel \"Y-axis\"\n", gnuplot); //グラフの Y 軸ラベル
72     fprintf_s(gnuplot,"set output \"%s/%s\"\n", directoryName, plotXYImageName); //名前をつけて保存
73     fputs("rep\n", gnuplot); //画像の保存の反映
74
75     //fputs("set view 90,360,1,1\n", gnuplot); //(X,Z) 平面
76     fprintf_s(gnuplot, "plot \"%s/%s\" u 2:4 with lp\n", directoryName, outputDataName); //データをプロット
77     fputs("set title \"Path X-Z\"\n", gnuplot); //グラフのタイトル
78     fputs("set xlabel \"X-axis\"\n", gnuplot); //グラフの X 軸ラベル
79     fputs("set ylabel \"Z-axis\"\n", gnuplot); //グラフの Y 軸ラベル
80     fprintf_s(gnuplot,"set output \"%s/%s\"\n", directoryName, plotXZImageName); //名前をつけて保存
81     fputs("rep\n", gnuplot); //画像の保存の反映
82
83     //fputs("set view 90,90,1,1\n", gnuplot); //(Y,Z) 平面
84     fprintf_s(gnuplot, "plot \"%s/%s\" u 3:4 with lp\n", directoryName,outputDataName); //データをプロット
```

```
85      fputs("set title \"Path Y-Z\"\n", gnuplot); //グラフのタイトル
86      fputs("set xlabel \"Y-axis\"\n", gnuplot); //グラフの X 軸ラベル
87      fputs("set ylabel \"Z-axis\"\n", gnuplot); //グラフの Y 軸ラベル
88      fprintf_s(gnuplot,"set output \"%s/%s\"\n", directoryName, plotYZImageName); //名前をつけて保存
89      fputs("rep\n", gnuplot); //画像の保存の反映
90
91      //確認用
92      //fprintf_s(gnuplot, "splot \"%s/%s\" with points\n", directoryName, outputDataName); //データをプロット
93      //fputs("pause 3\n", gnuplot); //プロットした結果をしばらく表示
94
95      fputs("quit\n", gnuplot); //gnuplot を終了
96      fflush(gnuplot); //コマンドを gnuplot で実行
97
98      return;
99  }
```

3.3.3.6 void Drawing::plot3DRealTime (int *countDataNum*, outputData *outputData[OUTPUTDATA_MAX]*)

< リアルタイムに位置をプロットするために利用 (c43)

< ファイルまでのパス名 (c43)

< リアルタイムプロット用のデータファイル (c43)

Drawing.cpp の 199 行目に定義があります。

参照先 directoryName, NOC.

```
200 {
201     FILE *realtimeplot;
202     char pathName[NOC];
203     char* fileName = "realplot.dat";
204
205     sprintf_s(pathName, "%s/%s", directoryName, fileName); //データファイルの保存先を指定
206     fopen_s(&realtimeplot, pathName, "a"); //ファイルを追記モードで開く (c43)
207     fprintf_s(realtimeplot, "%f %f %f %f\n", outputData[countDataNum].totalTime, outputData[countDataNum].x
    , outputData[countDataNum].y, outputData[countDataNum].z); //ファイルへ出力 (c43)
208     fclose(realtimeplot); //ファイルを閉じる (c43)
209
210     //プロット開始
211     if (first == true){ //1 回目は描画の初期設定をする
212         fprintf_s(gpr, "set grid\n"); //グリッドを描画
213         //fprintf_s(gpr, "unset key\n"); //凡例を消す
214         fprintf_s(gpr, "set xlabel \"X-axis\"\n"); //X 軸のラベルを設定
215         fprintf_s(gpr, "set ylabel \"Y-axis\"\n"); //Y 軸のラベルを設定
216         fprintf_s(gpr, "set zlabel \"Z-axis\"\n"); //Z 軸のラベルを設定
217
218         if (plotMode == 1){ //(1) 中心固定用
219             fprintf_s(gpr, "set xrange [-10:10]\n");
220             fprintf_s(gpr, "set yrange [-20:0]\n");
221             fprintf_s(gpr, "set zrange [50:200]\n");
222         }
223         else if (plotMode == 2){ //(2) 縦移動用
224             fprintf_s(gpr, "set xrange [-10:10]\n");
225             fprintf_s(gpr, "set yrange [-10:10]\n");
226             fprintf_s(gpr, "set zrange [50:300]\n");
227         }
228         else if(plotMode == 3){ //(3) 横移動用
229             fprintf_s(gpr, "set xrange [-60:60]\n");
230             fprintf_s(gpr, "set yrange [-10:10]\n");
231             fprintf_s(gpr, "set zrange [150:250]\n");
232         }
233         else{ //(4) 自動縮尺
234             fprintf_s(gpr, "set autoscale\n"); //自動縮尺 (c47)
235         }
236
237         first = false;
238     }
239     fprintf_s(gpr, "splot \"%s\" using 2:3:4 with linespoints\n", pathName);
240     fflush(gpr);
241
242     return;
243 }
```

このクラス詳解は次のファイルから抽出されました:

- **Drawing.hpp**
- **Drawing.cpp**

3.4 EV3Control クラス

EV3 を制御するためのクラス

`#include <EV3Control.hpp>`

公開メンバ関数

- **EV3Control** ()

  コンストラクタ
- ～**EV3Control** ()

  デストラクタ
- void **set6DoFEV3** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud, Point3d centroid, **Attitude-Angle** attitude_angle)

  最小二乗法によって求めた平均座標と位置を*EV3* の制御のために構造体に格納する *(c80)*

公開変数類

- **DoF6d ev3_6dof**

  EV3 の *6* 自由度 *(c80)*

3.4.1 詳解

EV3 を制御するためのクラス

EV3Control.hpp の 19 行目に定義があります。

3.4.2 構築子と解体子

3.4.2.1 EV3Control::EV3Control ( )

コンストラクタ

メソッドEV3Control::EV3Control(). コンストラクタ

EV3Control.cpp の 16 行目に定義があります。

```
17 {
18     //コンストラクタはなし
19 }
```

3.4.2.2 EV3Control::～EV3Control ( )

デストラクタ

メソッドEV3Control::EV3Control(). コンストラクタ

EV3Control.cpp の 24 行目に定義があります。

```
25 {
26     //デストラクタはなし
27 }
```

3.4.3 関数詳解

3.4.3.1 void EV3Control::set6DoFEV3 (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*, Point3d *centroid*, **AttitudeAngle** *attitude_angle*)

最小二乗法によって求めた平均座標と位置をEV3 の制御のために構造体に格納する (c80)

メソッドEV3Control::set6DoFEV3(). 最小二乗法によって求めた平均座標と位置をEV3 の制御のために構造体に格納する

引数

pcl::PointCloud<pcl::PointXYZRGB>::Ptr	&inputPointCloud, Point3d centroid, **AttitudeAngle** (p. 5) attitude_angle

EV3Control.cpp の 33 行目に定義があります。

参照先 ev3_6dof, DoF::pitch, AttitudeAngle::pitch, DoF::roll, AttitudeAngle::roll, DoF::x, DoF::y, DoF::yaw, AttitudeAngle::yaw, DoF::z.

参照元 main().

```
34 {
35     //FILE *fp;
36     //char filepath_measuredata[NOC];
37     //sprintf_s(filepath_measuredata, "data/%s/measuredata.dat", directoryName);
38     //fopen_s(&fp, filepath_measuredata, "w");
39
40     ev3_6dof.x = centroid.x;
41     ev3_6dof.y = centroid.y;
42     ev3_6dof.z = centroid.z;
43     ev3_6dof.yaw = attitude_angle.yaw;
44     ev3_6dof.roll = attitude_angle.roll;
45     ev3_6dof.pitch = attitude_angle.pitch;
46     cout << "[X, Y, Z, Yaw, Roll, Pitch, PointCloudNum]" << endl;
47     cout << "[ " << ev3_6dof.x << ", " << ev3_6dof.y << ", " << ev3_6dof.z << ", " <<
       ev3_6dof.yaw << ", " << ev3_6dof.roll << ", " << ev3_6dof.pitch << ", " << inputPointCloud->size() << " ]"
       << endl;
48
49     //fprintf_s(fp, "[\tX\tY\tZ\tYaw\tRoll\tPitch\tPointCloudNum\t]\n");
50     //fprintf_s(fp, "%f\t%f\t%f\t%f\t%f\t%d\n", ev3_6dof.x, ev3_6dof.y, ev3_6dof.z, ev3_6dof.yaw,
       ev3_6dof.roll, ev3_6dof.pitch, inputPointCloud->size());
51
52     //fclose(fp);
53     return;
54 }
```

3.4.4 メンバ詳解

3.4.4.1 DoF6d EV3Control::ev3_6dof

EV3 の 6 自由度 (c80)

EV3Control.hpp の 28 行目に定義があります。

参照元 main(), set6DoFEV3().

このクラス詳解は次のファイルから抽出されました:

- **EV3Control.hpp**
- **EV3Control.cpp**

3.5 ImageProcessing クラス

画像処理用のクラス

`#include <ImageProcessing.hpp>`

公開メンバ関数

- **ImageProcessing** ()

  コンストラクタ
- ~**ImageProcessing** ()

  デストラクタ
- void **showImage** (string windowName, Mat &input_image)

  ウインドウの名前を引数に追加 *(c31)*. *Mat* の表示 *(c17)*
- void **showImageTogether** (Mat &image1, Mat &image2)

  2 つの画像を一緒に表示 *(c36)*
- void **showImageTogether** (Mat &image1, Mat &image2, Mat &image3)

  3 つの画像を一緒に表示 *(c36)*
- void **loadInternalCameraParameter** (const string cameraParamFile)

  カメラキャリブレーションによって得られたパラメータを適用する *(c54)*
- Mat **getUndistortionImage** (Mat &inputOriginalImage)

  キャリブレーションデータを用いて *Kinect* から取得した画像を補正する *(c71)*
- Mat **getBackgroundSubstractionBinImage** (Mat &current_image, Mat &backgound_gray_image)

  背景差分によって得られた二値画像 *(c75)*
- Mat **getUnitMask** (Mat &input_binimage)

公開変数類

- int **th**

  二値化するときの閾値 *(c82)*
- int **neighborhood**
- int **closing_times**

3.5.1 詳解

画像処理用のクラス

ImageProcessing.hpp の 20 行目に定義があります。

3.5.2 構築子と解体子

3.5.2.1 ImageProcessing::ImageProcessing ( )

コンストラクタ

メソッドImageProcessing::ImageProcessing(). コンストラクタ

ImageProcessing.cpp の 16 行目に定義があります。

参照先 closing_times, neighborhood, th.

```
17 {
18     //コンストラクタは今のところなし
19     th = 13; //閾値の初期値 (c82)
20     neighborhood = 2;
21     closing_times = 3;
22 }
```

3.5.2.2 ImageProcessing::~ImageProcessing ( )

デストラクタ

メソッドImageProcessing::~ImageProcessing(). デストラクタ

ImageProcessing.cpp の 27 行目に定義があります。

```
28 {
29     //デストラクタは今のところなし
30 }
```

3.5.3 関数詳解

3.5.3.1 Mat ImageProcessing::getBackgroundSubstractionBinImage (Mat & *current_image*, Mat & *background_gray_image*)

背景差分によって得られた二値画像 (c75)

メソッドImageProcessing::getBackgroundSubstractionBinImage(). 背景差分によって得られた二値画像 (c75)

引数

cv::Mat&	current_image, cv::Mat& background_image

戻り値

cv::Mat median_bin_image

< 現在のグレースケール画像 (c75)

< 背景差分画像 (c74)

< 背景差分画像の二値画像 (c75)

< 背景差分画像の二値画像を平滑化したもの (c75)

ImageProcessing.cpp の 126 行目に定義があります。

参照先 closing_times, neighborhood, th.

参照元 main().

```
127 {
128     Mat current_gray_image;
129     Mat diff_gray_image;
130     Mat diff_bin_image;
131     Mat median_bin_image;
132     //Mat opening_image; //(仮)
133     Mat closing_image;
134
135     cvtColor(current_image, current_gray_image, CV_BGR2GRAY); //現フレームの画像をグレースケールに
136     absdiff(current_gray_image, background_gray_image, diff_gray_image); //差分画像取得
137     //showImage("差分画像", diff_gray_image);
138
139     //EV3 用
140     //threshold(diff_gray_image, diff_bin_image, /*13*/20, 255, THRESH_BINARY); //二値化
142     //medianBlur(diff_bin_image, median_bin_image, 7); //ノイズ除去
145     //morphologyEx(median_bin_image, closing_image, MORPH_CLOSE, Mat(), Point(-1, -1), 7);
      //クロージング（膨張→収縮）処理．穴埋めに使われる
147
148
149     threshold(diff_gray_image, diff_bin_image, th, 255, THRESH_BINARY); //二値化
150     //showImage("二値画像", diff_bin_image);
151     medianBlur(diff_bin_image, median_bin_image, 2*neighborhood+1); //ノイズ除去
152     //showImage("平滑化処理後", median_bin_image);
153     //morphologyEx(median_bin_image, opening_image, MORPH_OPEN, Mat(), Point(-1, -1), 5); //オープニング（縮
小→膨張）処理
154     morphologyEx(median_bin_image, closing_image, MORPH_CLOSE, Mat(), Point(-1, -1), 2*
      closing_times+1); //クロージング（膨張→収縮）処理．穴埋めに使われる
155     //showImage("穴埋め処理後", closing_image);
156     return closing_image;
157     //return median_bin_image;
158 }
```

3.5.3.2 Mat ImageProcessing::getUndistortionImage (Mat & *inputOriginalImage*)

キャリブレーションデータを用いてKinect から取得した画像を補正する (c71)

メソッドImageProcessing::getUndistortionImage(). キャリブレーションデータを用いてKinect から取得した画像を補正する

引数

cv::Mat&	inputOriginalImage

戻り値

cv::Mat undistortionImage

ImageProcessing.cpp の 112 行目に定義があります。

参照元 main().

```
113 {
114     Mat undistotionImage;
115
116     undistort(inputOriginalImage, undistotionImage, internal_cameraparam, distortion_coefficients, Mat());
117
118     return undistotionImage;
119 }
```

3.5.3.3 Mat ImageProcessing::getUnitMask (Mat & *input_binimage*)

ImageProcessing.cpp の 160 行目に定義があります。

```
161 {
162     int x_min;
163     int x_border1;
164     int x_border2;
165     int x_max;
166     int y_min;
167     int y_border;
168     int y_max;
169
170     bool ymin_check = false; //y_min が見つかっていないとき
171
172     //x の最大値と最小値の計測
173     x_min = input_binimage.cols;
174     x_max = 0;
175     //y の最大値と最小値の計測
176     y_min = 0;
177     y_max = 0;
178
179     //x と y の最大値と最小値を探す
180     for (int y = 0; y < input_binimage.rows;y++){
181         for (int x = 0; x < input_binimage.cols;x++){
182             if (input_binimage.at<unsigned char>(y, x) == 255) //白ピクセルなら特定の処理を行う
183             {
184                 if (x_min > x){
185                     x_min = x;
186                 }
187                 if (x_max < x){
188                     x_max = x;
189                 }
190                 if (ymin_check == false){ //y_min がみつかっていなければ. 一度しか実行されない
191                     y_min = y; //そのときの y を保存
192                     ymin_check = true; //フラグを true にする
193                 }
194                 if (y_max < y){ //現代の最大より新しい y が大きければ
195                     y_max = y; //新しい y を最大値とする
196                 }
197             }
198         }
199     }
200     cout << "x_min => " << x_min << ", x_max => " << x_max << ", y_min => " << y_min << ", y_max => " <<
    y_max << endl;
201
202     //x 方向の切り取り
203     x_border1 = (x_min + x_max) * /*0.18*/0.17;
```

```
204     x_border2 = (x_min + x_max) * /*0.82*/0.83;
205     //左部を白にする
206     /*for (int y = 0; y < input_binimage.rows; y++){
207         for (int x = x_min; x < x_border1; x++){
208             if (input_binimage.at<unsigned char>(y, x) == 255){
209                 input_binimage.at<unsigned char>(y, x) = 0;
210             }
211         }
212     }
213     //右部を白にする
214     for (int y = 0; y < input_binimage.rows; y++){
215         for (int x = x_border2; x < x_max; x++){
216             if (input_binimage.at<unsigned char>(y, x) == 255){
217                 input_binimage.at<unsigned char>(y, x) = 0;
218             }
219         }
220     }*/
221     //imwrite("before_cut.jpg", input_binimage);
222     //上限と下限からカットするボーダーを決定する
223     y_border = (y_max + y_min) * /*0.45*/0.47; //影の影響で y_max が増えるため少し大きめに設定するのが良い
224     //下部を白にする
225     int step;
226     step = 30;
227     for (int y = y_min; y < y_min + step; y++){
228         for (int x = 0; x < input_binimage.cols; x++){
229             if (input_binimage.at<unsigned char>(y, x) == 255){
230                 input_binimage.at<unsigned char>(y, x) = 0;
231             }
232         }
233     }
234
235     //下部の切り取り
236     for (int y = y_border; y <= y_max; y++){
237         for (int x = 0; x < input_binimage.cols; x++){
238             if (input_binimage.at<unsigned char>(y, x) == 255){
239                 input_binimage.at<unsigned char>(y, x) = 0;
240             }
241         }
242     }
243
244     //showImage("TEST", input_binimage);
245     return input_binimage;
246 }
```

3.5.3.4 void ImageProcessing::loadInternalCameraParameter (const string *cameraParamFile*)

カメラキャリブレーションによって得られたパラメータを適用する (c54)

ImageProcessing::loadInternalCameraParam(). カメラキャリブレーションによって得られたカメラパラメータを適用するメソッド (c54)

引数

const	string cameraParamFile

ImageProcessing.cpp の 95 行目に定義があります。

参照元 main().

```
96 {
97     //xml ファイルの読み込み
98     FileStorage fs(cameraParamFile, FileStorage::READ); //読み込みモード
99     //内部パラメータの読み込み
100     fs["camera_matrix"] >> internal_cameraparam; //内部パラメータを読み込む
101     fs["distortion_coefficients"] >> distortion_coefficients; //歪み係数を読み込む
102
103     cout << "loaded " << cameraParamFile << ". " << endl;
104     return;
105 }
```

3.5.3.5 void ImageProcessing::showImage (string *windowName*, Mat & *input_image*)

ウインドウの名前を引数に追加 (c31). Mat の表示 (c17)

メソッドImageProcessing::showImage().cv::Mat を表示

引数

std::string	windowName, cv::Mat& input_image

ImageProcessing.cpp の 36 行目に定義があります。

参照元 main(), showImageTogether().

```
37 {
38     namedWindow(windowName, CV_WINDOW_AUTOSIZE | CV_WINDOW_FREERATIO);
39     imshow(windowName, input_image);
40     return;
41 }
```

3.5.3.6 void ImageProcessing::showImageTogether (Mat & *image1*, Mat & *image2*)

2 つの画像を一緒に表示 (c36)

メソッドImageProcessing::showTogetherImage().2 つの cv::Mat を 1 つのウインドウに表示

引数

cv::Mat&	image1, cv::Mat& image2

ImageProcessing.cpp の 47 行目に定義があります。

参照先 showImage().

```
48 {
49     //ウインドウ名と合成画像を定義
50     char* window_name = "処理結果";
51     int win_width = image1.cols + image2.cols;
52     int win_height = max(image1.rows, image2.rows); //ウインドウの高さは高い方に合わせる
53     Mat all_img(Size(win_width, win_height), CV_8UC3);
54
55     //画像を合成
56     Mat roi1(all_img, Rect(0, 0, image1.cols, image1.rows));
57     Mat roi2(all_img, Rect(image1.cols, 0, image2.cols, image2.rows));
58     image1.copyTo(roi1);
59     image2.copyTo(roi2);
60
61     showImage(window_name, all_img); //合成した画像を表示
62
63     return;
64 }
```

3.5.3.7 void ImageProcessing::showImageTogether (Mat & *image1*, Mat & *image2*, Mat & *image3*)

3 つの画像を一緒に表示 (c36)

メソッドImageProcessing::showTogetherImage().3 つの cv::Mat を 1 つのウインドウに表示

引数

cv::Mat&	image1, cv::Mat& image2, cv::Mat& image3

ImageProcessing.cpp の 70 行目に定義があります。

参照先 showImage().

```
71 {
72     //ウインドウ名と合成画像を定義
73     char* window_name = "処理結果";
74     int win_width = image1.cols + image2.cols + image3.cols;
75     int win_height = max({ image1.rows, image2.rows, image3.rows }); //ウインドウの高さは高い方に合わせる
76     Mat all_img(Size(win_width, win_height), CV_8UC3);
77
78     //画像を合成
79     Mat roi1(all_img, Rect(0, 0, image1.cols, image1.rows));
80     Mat roi2(all_img, Rect(image1.cols, 0, image2.cols, image2.rows));
81     Mat roi3(all_img, Rect(image1.cols + image2.cols, 0, image3.cols, image3.rows));
```

```
82      image1.copyTo(roi1);
83      image2.copyTo(roi2);
84      image3.copyTo(roi3);
85
86      showImage(window_name, all_img); //合成した画像を表示
87
88      return;
89  }
```

3.5.4 メンバ詳解

3.5.4.1 int ImageProcessing::closing_times

ImageProcessing.hpp の 41 行目に定義があります。

参照元 getBackgroundSubstractionBinImage(), ImageProcessing(), main().

3.5.4.2 int ImageProcessing::neighborhood

ImageProcessing.hpp の 40 行目に定義があります。

参照元 getBackgroundSubstractionBinImage(), ImageProcessing(), main().

3.5.4.3 int ImageProcessing::th

二値化するときの閾値 (c82)

ImageProcessing.hpp の 39 行目に定義があります。

参照元 getBackgroundSubstractionBinImage(), ImageProcessing(), main().

このクラス詳解は次のファイルから抽出されました:

- **ImageProcessing.hpp**
- **ImageProcessing.cpp**

3.6 Kinect クラス

Kinect 操作用のクラス

`#include <Kinect.hpp>`

公開メンバ関数

- **Kinect** ()
  コンストラクタ
- ~**Kinect** ()
  デストラクタ
- void **initialize** ()
  Kinect の初期化
- void **createInstance** ()
  インスタンスの生成
- Mat **drawRGBImage** (Mat &**image**)
  RGB カメラの処理
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **getPointCloud** (Mat &Mat_image)
  Depth カメラの処理 *(c57)*

- int **getDistance** (Mat &**image**)
  距離を取得 *(c49)*

公開変数類

- HANDLE **streamEvent**
  RGB,Depth カメラのフレーム更新イベントを待つためのイベントハンドル
- int **key**
  ウィンドウ表示のウェイトタイム格納変数
- int **actualExtractedNum**

3.6.1 詳解

Kinect 操作用のクラス

Kinect.hpp の 31 行目に定義があります。

3.6.2 構築子と解体子

3.6.2.1 Kinect::Kinect ( )

コンストラクタ

メソッドKinect::Kinect(). コンストラクタ

Kinect.cpp の 16 行目に定義があります。

```
17 {
18     countKinect = 0; //Kinect の数を初期化
19 }
```

3.6.2.2 Kinect::∼Kinect ( )

デストラクタ

メソッドKinect::∼Kinect(). デストラクタ

Kinect.cpp の 24 行目に定義があります。

```
25 {
26     //終了処理
27     if (kinect != 0){
28         kinect->NuiShutdown();
29         kinect->Release();
30     }
31 }
```

3.6.3 関数詳解

3.6.3.1 void Kinect::createInstance ( )

インスタンスの生成

メソッドKinect::createInstance(). インスタンスの生成

Kinect.cpp の 36 行目に定義があります。

参照先 ERROR_CHECK.

参照元 initialize().

```
37 {
38     //接続されている Kinect の数を取得する
39     ERROR_CHECK(::NuiGetSensorCount(&countKinect));
40     if (countKinect == 0){
41         throw runtime_error("Please Connect the Kinect.");
42     }
43
44     ERROR_CHECK(::NuiCreateSensorByIndex(0, &kinect)); //最初の Kinect のインスタンスを生成する
45
46     //Kinect の状態を取得する
47     HRESULT status = kinect->NuiStatus();
48     if (status != S_OK){
49         throw runtime_error("You Cannot Use the Kinect.");
50
51     }
52
53     return;
54 }
```

3.6.3.2 Mat Kinect::drawRGBImage (Mat & *image*)

RGB カメラの処理

メソッドKinect::drawRGBImage(Mat& image).RGB カメラの処理

引数

cv::Mat&	image

戻り値

cv::Mat image

Kinect.cpp の 82 行目に定義があります。

参照先 ERROR_CHECK.

参照元 main().

```
83 {
84     try{
85         //RGB カメラのフレームデータを取得する
86         NUI_IMAGE_FRAME imageFrame = { 0 };
87         ERROR_CHECK(kinect->NuiImageStreamGetNextFrame(imageStreamHandle, 0, &imageFrame));
88
89         //画像データの取得
90         NUI_LOCKED_RECT colorData;
91         imageFrame.pFrameTexture->LockRect(0, &colorData, 0, 0);
92
93         //画像データのコピー
94         image = Mat(height, width, CV_8UC4, colorData.pBits);
95
96         //フレームデータの解放
97         ERROR_CHECK(kinect->NuiImageStreamReleaseFrame(imageStreamHandle, &imageFrame));
98     }
99     catch (exception& ex){ //例外処理 (c57)
100         cout << ex.what() << endl;
101     }
102     return (image); //RGB カメラから画像を取得し返す (c30)
103 }
```

3.6.3.3 int Kinect::getDistance (Mat & *image*)

距離を取得 (c49)

3.6.3.4 pcl::PointCloud< pcl::PointXYZRGB >::Ptr Kinect::getPointCloud (Mat & *Mat_image*)

Depth カメラの処理 (c57)

Kinect.cpp の 110 行目に定義があります。

参照先 CAMERA_RESOLUTION, ERROR_CHECK, image.

参照元 main().

```
111 {
112     try{
113         pcl::PointCloud<pcl::PointXYZRGB>::Ptr points(new pcl::PointCloud<pcl::PointXYZRGB>()); //
    ポイントクラウド保存用 (c57)
114         points->width = width;
115         points->height = height;
116 
117         //距離カメラのフレームデータを取得
118         NUI_IMAGE_FRAME depthFrame = { 0 };
119         ERROR_CHECK(kinect->NuiImageStreamGetNextFrame(depthStreamHandle, 0, &depthFrame));
120 
121         //距離データを取得する
122         NUI_LOCKED_RECT depthData = { 0 };
123         depthFrame.pFrameTexture->LockRect(0, &depthData, 0, 0);
124 
125         USHORT* depth = (USHORT*)depthData.pBits;
126         for (int i = 0; i < (depthData.size / sizeof(USHORT)); ++i){
127 
128             USHORT distance = ::NuiDepthPixelToDepth(depth[i]);
129 
130             //USHORT player = ::NuiDepthPixelToPlayerIndex(depth[i]);
131             LONG depthX = i % width;
132             LONG depthY = i / width;
133             LONG colorX = depthX;
134             LONG colorY = depthY;
135 
136             // 距離カメラの座標を、RGB カメラの座標に変換する
137             kinect->NuiImageGetColorPixelCoordinatesFromDepthPixelAtResolution(
    CAMERA_RESOLUTION, CAMERA_RESOLUTION, 0, depthX, depthY, 0/*depth[i]*/, &colorX, &colorY);
138 
139             //点群取得処理．渡された差分画像に応じて条件を入れ替える
140             //Vector4 real = NuiTransformDepthImageToSkeleton(depthX, depthY, distance, CAMERA_RESOLUTION);
141             Vector4 real = NuiTransformDepthImageToSkeleton(depthX, depthY, distance << 3,
    CAMERA_RESOLUTION); //左に 3 ビットすることでプレーヤー情報を含む深度データを渡し，座標を変換する
142             if (Mat_image.at<UCHAR>(colorY, colorX) == 255){ //二値画像に対して点群を抽出する際はこっち (白色の点群
を抽出) (c70)
143                 pcl::PointXYZRGB point; //点群用の変数を確保
144                 point.x = real.x*1000.0f; //ポイントクラウドの x 座標を格納 [mm]
145                 point.y = real.y*1000.0f; //ポイントクラウドの y 座標を格納 [mm]
146                 point.z = real.z*1000.0f; //ポイントクラウドの z 座標を格納 [mm]
147 
148                 //cout << point << endl;
149                 //テクスチャ(その座標の色を格納していく)
150                 Vec4b color = image.at<Vec4b>(colorY, colorX); //色格納用の変数
151                 point.r = color[2]; //赤要素
152                 point.g = color[1]; //緑要素
153                 point.b = color[0]; //青要素
154                 points->push_back(point); //点群を出力
155             }
156         }
157         cloud = points; //点群をコピー
158         //フレームデータを開放する (c58)
159         ERROR_CHECK(kinect->NuiImageStreamReleaseFrame(depthStreamHandle, &depthFrame));
160     }
161     catch (exception& ex){
162         cout << ex.what() << endl;
163     }
164     return cloud;
165 }
```

3.6.3.5 void Kinect::initialize ( )

Kinect の初期化

メソッドKinect::initialize().Kinect の初期化

Kinect.cpp の 59 行目に定義があります。

参照先 CAMERA_RESOLUTION, createInstance(), ERROR_CHECK, streamEvent.

参照元 main().

```
60 {
61     createInstance(); //createInstance() の処理へ以降
62 
63     ERROR_CHECK(kinect->NuiInitialize(NUI_INITIALIZE_FLAG_USES_COLOR | NUI_INITIALIZE_FLAG_USES_DEPTH)); //
```

```
    Kinect の設定を初期化
64   ERROR_CHECK(kinect->NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR,
    CAMERA_RESOLUTION, 0, 2, 0, &imageStreamHandle)); //RGB カメラを初期化
65   ERROR_CHECK(kinect->NuiImageStreamOpen(NUI_IMAGE_TYPE_DEPTH,
    CAMERA_RESOLUTION, 0, 2, 0, &depthStreamHandle)); //Depth カメラを初期化
66   ERROR_CHECK(kinect->NuiImageStreamSetImageFrameFlags(depthStreamHandle,
    NUI_IMAGE_STREAM_FLAG_ENABLE_NEAR_MODE)); //Near モード
67
68   //フレーム更新のイベントハンドルを作成
69   streamEvent = ::CreateEvent(0, TRUE, FALSE, 0);
70   ERROR_CHECK(kinect->NuiSetFrameEndEvent(streamEvent, 0));
71
72   ::NuiImageResolutionToSize(CAMERA_RESOLUTION, width, height); //指定した解像度の画面サイズを取得する
73
74   return;
75 }
```

3.6.4 メンバ詳解

3.6.4.1 int Kinect::actualExtractedNum

Kinect.hpp の 55 行目に定義があります。

3.6.4.2 int Kinect::key

ウィンドウ表示のウェイトタイム格納変数

Kinect.hpp の 54 行目に定義があります。

参照元 main().

3.6.4.3 HANDLE Kinect::streamEvent

RGB,Depth カメラのフレーム更新イベントを待つためのイベントハンドル

Kinect.hpp の 53 行目に定義があります。

参照元 initialize(), main().

このクラス詳解は次のファイルから抽出されました:

- **Kinect.hpp**
- **Kinect.cpp**

3.7 LeastSquareMethod クラス

最小二乗法を行うクラス

`#include <LeastSquareMethod.hpp>`

公開メンバ関数

- **LeastSquareMethod** ()
  コンストラクタ
- ~**LeastSquareMethod** ()
  デストラクタ
- Eigen::Vector3f **getCoefficient** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud)
  最小二乗法によって平面 *ax+by+c=0* の係数 *[a b c]'* を求めるメソッド
- **AttitudeAngle3d calcYawRollPitch** (Eigen::Vector3f coefficient_plane)
  最小二乗法によって求めた *[a b c]'* を用いて平面の姿勢を計算する

3.7.1 詳解

最小二乗法を行うクラス

LeastSquareMethod.hpp の 20 行目に定義があります。

3.7.2 構築子と解体子

3.7.2.1 LeastSquareMethod::LeastSquareMethod ( )

コンストラクタ

メソッドLeastSquareMethod::LeastSquareMethod(). コンストラクタ

LeastSquareMethod.cpp の 16 行目に定義があります。

```
17 {
18     //繧^^ef^^bd^^b3繝^^ef^^bd^^b3繧^^ef^^bd^^b9繝医Λ繧^^ef^^bd^^af繧^^ef^^bd^^bf
19 }
```

3.7.2.2 LeastSquareMethod::～LeastSquareMethod ( )

デストラクタ

メソッドLeastSquareMethod::～LeastSquareMethod(). デストラクタ

LeastSquareMethod.cpp の 24 行目に定義があります。

```
25 {
26     //デストラクタ
27 }
```

3.7.3 関数詳解

3.7.3.1 AttitudeAngle3d LeastSquareMethod::calcYawRollPitch (Eigen::Vector3f *coefficient_plane*)

最小二乗法によって求めた [a b c]' を用いて平面の姿勢を計算する

メソッドLeastSquareMethod::calcYawRollPitch(). 最小二乗法によって求めた [a b c]' を用いて平面の姿勢を計算する

LeastSquareMethod.cpp の 77 行目に定義があります。

参照先 AttitudeAngle::pitch, AttitudeAngle::roll, AttitudeAngle::yaw.

参照元 main().

```
78 {
79     AttitudeAngle3d attitude_angle_rad; //姿勢角（ラジアン）
80     AttitudeAngle3d attitude_angle_deg; //姿勢角（度）
81
82     //ヨー角の計算
83     attitude_angle_rad.yaw = -atan(coefficient_plane.x());
84     attitude_angle_deg.yaw = attitude_angle_rad.yaw / M_PI * 180.0;
85
86     //ロール角の計算
87     attitude_angle_rad.roll = atan2(-coefficient_plane.x(), coefficient_plane.y());
88     attitude_angle_deg.roll = attitude_angle_rad.roll / M_PI * 180.0;
89
90     //ピッチ角の計算
91     attitude_angle_rad.pitch = atan2(1, coefficient_plane.y());
92     attitude_angle_deg.pitch = attitude_angle_rad.pitch / M_PI * 180.0;
93
94     return attitude_angle_deg;
95 }
```

3.7.3.2 Eigen::Vector3f LeastSquareMethod::getCoefficient (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

最小二乗法によって平面 ax+by+c=0 の係数 [a b c]' を求めるメソッド

メソッドLeastSquareMethod::getCoefficient(). 最小二乗法によって平面 ax+by+c=0 の係数 [a b c]' を求めるメソッド

引数

pcl::PointCloud<pcl::PointXYARGB>::Ptr	&inputPointCloud

戻り値

Eigen::Vector3f coefficient_plane

LeastSquareMethod.cpp の 34 行目に定義があります。

参照元 main().

```
35 {
36     Eigen::Vector3f coefficient_plane(0.0, 0.0, 0.0);
37     Eigen::Matrix3f m1;
38     m1.Zero();
39     Eigen::Vector3f m2(0.0, 0.0, 0.0);
40
41     //3 × 3 行列 S=[Sxx Sxy Sx;Sxy Syy Sy;Sx Sy inputPointCloud->size()] の初期化
42     double Szz = 0, Sxz = 0, Syz = 0, Sz = 0, Sxx = 0, Sxy = 0, Sx = 0, Syy = 0, Sy = 0;
43     //最小二乗法によって求めた A=[a b c]' の要素の初期化
44     //double a = 0, b = 0, c = 0;
45     //cout << "input Size => " << inputPointCloud->size() << endl;
46     for (int i = 1; i < inputPointCloud->size(); i++){ //それぞれの要素の計算
47         Szz = Szz + inputPointCloud->points[i].z * inputPointCloud->points[i].z;
48         Sxz = Sxz + inputPointCloud->points[i].x * inputPointCloud->points[i].z;
49         Syz = Syz + inputPointCloud->points[i].y * inputPointCloud->points[i].z;
50         Sz = Sz + inputPointCloud->points[i].z;
51         Sxx = Sxx + inputPointCloud->points[i].x * inputPointCloud->points[i].x;
52         Sxy = Sxy + inputPointCloud->points[i].x * inputPointCloud->points[i].y;
53         Sx = Sx + inputPointCloud->points[i].x;
54         Syy = Syy + inputPointCloud->points[i].y * inputPointCloud->points[i].y;
55         Sy = Sy + inputPointCloud->points[i].y;
56     }
57
58     m1 << Sxx, Sxy, Sx,
59         Sxy, Syy, Sy,
60         Sx, Sy, inputPointCloud->size();
61     //cout << "m1 => \n" << m1 << endl;
62     //cout << "m1_inverse => \n" << m1.inverse() << endl;
63
64     m2 << Sxz, Syz, Sz;
65     //cout << "m2 => \n" << m2 << endl;
66
67     coefficient_plane = m1.inverse() * m2;
68     //cout << "coefficient_plane => \n" << coefficient_plane << endl;
69
70     return coefficient_plane;
71 }
```

このクラス詳解は次のファイルから抽出されました:

- **LeastSquareMethod.hpp**
- **LeastSquareMethod.cpp**

3.8 outputData 構造体

```
#include <PathTrackingAndInductionOfTheRobot.hpp>
```

公開変数類

- double **totalTime**
- float **x**
- float **y**
- float **z**

3.8.1 詳解

PathTrackingAndInductionOfTheRobot.hpp の 21 行目に定義があります。

3.8.2 メンバ詳解

3.8.2.1 double outputData::totalTime

PathTrackingAndInductionOfTheRobot.hpp の 22 行目に定義があります。

3.8.2.2 float outputData::x

PathTrackingAndInductionOfTheRobot.hpp の 23 行目に定義があります。

3.8.2.3 float outputData::y

PathTrackingAndInductionOfTheRobot.hpp の 24 行目に定義があります。

3.8.2.4 float outputData::z

PathTrackingAndInductionOfTheRobot.hpp の 25 行目に定義があります。
この構造体詳解は次のファイルから抽出されました:

- **PathTrackingAndInductionOfTheRobot.hpp**

3.9 Point3 構造体

```
#include <PathTrackingAndInductionOfTheRobot.hpp>
```

公開変数類

- int **x**
- int **y**
- USHORT **z**

3.9.1 詳解

PathTrackingAndInductionOfTheRobot.hpp の 15 行目に定義があります。

3.9.2 メンバ詳解

3.9.2.1 int Point3::x

PathTrackingAndInductionOfTheRobot.hpp の 16 行目に定義があります。

3.9.2.2 int Point3::y

PathTrackingAndInductionOfTheRobot.hpp の 17 行目に定義があります。

3.9.2.3 USHORT Point3::z

PathTrackingAndInductionOfTheRobot.hpp の 18 行目に定義があります。

この構造体詳解は次のファイルから抽出されました:

- **PathTrackingAndInductionOfTheRobot.hpp**

3.10 PointCloudLibrary クラス

点群処理を行うクラス

`#include <PointCloudLibrary.hpp>`

公開メンバ関数

- **PointCloudLibrary** ()

  メソッド*PointCloudLibrary::PointCloudLibrary().* コンストラクタ
- **PointCloudLibrary** (bool passthroughflag, bool downsamplingflag, bool statisticaloutlierremovalflag, bool mlsflag, bool extractplaneflag)

  メソッド*PointCloudLibrary::PointCloudLibrary().* コンストラクタ *(c64)*
- ∼**PointCloudLibrary** ()

  メソッド*PointCloudLibrary::∼PointCloudLibrary().* デストラクタ
- void **initializePointCloudViewer** (string cloudviewer_name)

  ポイントクラウドビューアーを初期化
- void **initializePCLVisualizer** (string pclvisualizer_name)
- void **loadPLY** (char ∗ply_name)

  メソッド*PointCloudLibrary::loadPLY().* *ply* ファイルを読み込む
- void **flagChecker** ()

  メソッド*PointCloudLibrary::flagChecker().PCL* 処理に関する処理の有無を判定するフラグ変数を反転させるメソッド *(c64)*
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **passThroughFilter** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud, char ∗axis, float min, float max)

  パススルーフィルタ．*z* の値の距離に応じてカット可能
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **removeOutlier** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud)

  メソッド*PointCloudLibrary::removeOutlier().* 外れ値を除去するメソッド *(c59)*
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **radiusOutlierRemoval** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud)

  メソッド*PointCloudLibrary::radiusOutlierRemoval().* 外れ値を除去するメソッド *(c60)*

- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **downSamplingUsingVoxelGridFilter** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud, float leafSizeX, float leafSizeY, float leafSizeZ)
  ダウンサンプリングの
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **smoothingUsingMovingLeastSquare** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud, bool compute_normals, bool polynomial_fit, double radius)
  スムージング
- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **getExtractPlaneAndClustering** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud, bool optimize, int maxIterations, double threshold, bool negative1, bool negative2, double tolerance, int minClusterSize, int maxClusterSize)
  平面検出とクラスタリング
- Point3d **getCentroidCoordinate3d** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud)
  メソッド *getCentroidCoordinate*
- pcl::PointCloud< pcl::Normal >::Ptr **getSurfaceNormals** (pcl::PointCloud< pcl::PointXYZRGB >::Ptr &inputPointCloud)
  法線を計算する

公開変数類

- pcl::PointCloud
  < pcl::PointXYZRGB >::Ptr **model**
- Point3d **centroid**
  平均座標
- pcl::visualization::CloudViewer * **viewer**
- pcl::visualization::PCLVisualizer * **visualizer**
- bool **passthrough_flag**
- bool **downsampling_flag**
- bool **statisticaloutlierremoval_flag**
- bool **mls_flag**
- bool **extractplane_flag**

3.10.1 詳解

点群処理を行うクラス

PointCloudLibrary.hpp の 20 行目に定義があります。

3.10.2 構築子と解体子

3.10.2.1 PointCloudLibrary::PointCloudLibrary ( )

メソッドPointCloudLibrary::PointCloudLibrary(). コンストラクタ

PointCloudLibrary.cpp の 16 行目に定義があります。

```
17 {
18     //繧^^ef^^bd^^b3繝^^ef^^bd^^b3繧^^ef^^bd^^b9繝医Λ繧^^ef^^bd^^af繧^^ef^^bd^^bf
19 }
```

3.10.2.2 PointCloudLibrary::PointCloudLibrary (bool *passthroughflag,* bool *downsamplingflag,* bool *statisticaloutlierremovalflag,* bool *mlsflag,* bool *extractplaneflag*)

メソッドPointCloudLibrary::PointCloudLibrary(). コンストラクタ (c64)

引数

bool	flag_removeOutlier, bool flag_downsampling, bool flag_MLS, bool flag_extractPlane

PointCloudLibrary.cpp の 25 行目に定義があります。

参照先 downsampling_flag, extractplane_flag, mls_flag, passthrough_flag, statisticaloutlierremoval_flag.

```
26 {
27     //繧^^ef^^bd^^b3繝^^ef^^bd^^b3繧^^ef^^bd^^b9繝医Λ繧^^ef^^bd^^af繧^^ef^^bd^^bf
28     passthrough_flag = passthroughflag;
29     downsampling_flag = downsamplingflag;
30     statisticaloutlierremoval_flag = statisticaloutlierremovalflag;
31     mls_flag = mlsflag;
32     extractplane_flag= extractplaneflag;
33 }
```

3.10.2.3 PointCloudLibrary::~PointCloudLibrary ( )

メソッドPointCloudLibrary::~PointCloudLibrary(). デストラクタ

PointCloudLibrary.cpp の 38 行目に定義があります。

```
39 {
40     //デストラクタ
41 }
```

3.10.3 関数詳解

3.10.3.1 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::downSamplingUsingVoxelGridFilter (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud,* float *leafSizeX,* float *leafSizeY,* float *leafSizeZ*)

ダウンサンプリングの

メソッドPointCloudLibrary::downSamplingUsingVoxelGridFilter(). ダウンサンプリング処理を行うメソッド (c59)

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 167 行目に定義があります。

参照元 main().

```
168 {
169     cout << "before\tVoxel Grid Filter\t=>\t" << inputPointCloud->size() << endl;
170
171     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>()); //
      フィルタリング後用のポイントクラウドを宣言
172     pcl::VoxelGrid<pcl::PointXYZRGB> vg;
173
174     vg.setInputCloud(inputPointCloud);
175     //sor.setLeafSize() でダウンサンプリングの程度を変更
176     vg.setLeafSize(leafSizeX, leafSizeY, leafSizeZ);
177     vg.filter(*filtered);
178
179     //ポイントクラウドをしっかり保持できているかサイズを確認
180     cout << "after\tVoxel Grid Filter\t=>\t" << filtered->size() << endl; //出力されるポイントクラウドのサイズ
181     return filtered;
182 }
```

3.10.3.2 void PointCloudLibrary::flagChecker ( )

メソッドPointCloudLibrary::flagChecker().PCL 処理に関する処理の有無を判定するフラグ変数を反転させるメソッド (c64)

PointCloudLibrary.cpp の 77 行目に定義があります。

参照先 downsampling_flag, extractplane_flag, mls_flag, passthrough_flag, statisticaloutlierremoval_flag.

参照元 main().

```
78 {
79     if (GetAsyncKeyState('X')){ //X が入力されたので、パススルーフィルターのフラグを反転
80         passthrough_flag = !passthrough_flag;
81     }
82     if (GetAsyncKeyState('C')){ //C が入力されたので、ダウンサンプリング処理のフラグを反転
83         downsampling_flag = !downsampling_flag;
84     }
85     if (GetAsyncKeyState('V')){ //V が入力されたので、統計的な外れ値除去処理のフラグを反転
86         statisticaloutlierremoval_flag = !statisticaloutlierremoval_flag;
87     }
88     if (GetAsyncKeyState('B')){  //N が入力されたので、MLS 処理のフラグを反転
89         mls_flag = !mls_flag;
90     }
91     if (GetAsyncKeyState('N')){ //M が入力されたので、平面検出のフラグを反転
92         extractplane_flag = !extractplane_flag;
93     }
94
95     cout << "範囲外除去 (X) => " << passthrough_flag << " ダウンサンプリング (C) => " <<
      downsampling_flag << " 外れ値 (V) => " << statisticaloutlierremoval_flag << " MLS(B) => " <<
      mls_flag << " 平面検出 (N) => " << extractplane_flag << endl;
96     return;
97 }
```

3.10.3.3 Point3d PointCloudLibrary::getCentroidCoordinate3d (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

メソッド getCentroidCoordinate

引数

pcl::PointCloud<pcl::PointXYZRGB>::Ptr	&inputPointCloud

戻り値

Point3f centroid

PointCloudLibrary.cpp の 316 行目に定義があります。

参照元 main().

```
317 {
318     //cout << "inputPointCloud => " << inputPointCloud->size() << endl; //最大のクラスタを受け取れているか確認
(c76)
319     //FILE *pointcloud; //最終 1 フレーム分. gnuplot で表示するために点群をファイルに出力する用
320     //FILE *centroid; //最終 1 フレーム分. gnuplot で表示するために平均座標 (重心) をファイルに出力する用
321     Point3d centroid_coordinate = 0; //重心座標
322     Point3d sum_pointcloud = 0; //座標の合計
323
324     //char filepath_pointcloud[NOC];
325     //char filepath_centroid[NOC];
326     //sprintf_s(filepath_pointcloud, "data/%s/pointcloud.dat", directoryName);
327     //sprintf_s(filepath_centroid, "data/%s/centroid.dat", directoryName);
328
329     //ファイルオープン gnuplot の確認用
330     //fopen_s(&pointcloud, filepath_pointcloud, "w"); //
331     //fopen_s(&centroid, filepath_centroid, "w");
332
333     //summention coordinate. ※ inputPointCloud->width == inputPointCloud->size().
```

```
334     for (int i = 1; i < inputPointCloud->size(); i++){
335         //cout << i << " : " << "[x, y, z] => [ " << inputPointCloud->points[i].x << ", " <<
    inputPointCloud->points[i].y << ", " << inputPointCloud->points[i].z << " ] " << endl;
336         //fprintf_s(pointcloud, "%f %f %f\n", inputPointCloud->points[i].x, inputPointCloud->points[i].y,
    inputPointCloud->points[i].z); //ファイルに出力
337         sum_pointcloud.x = sum_pointcloud.x + inputPointCloud->points[i].x; //点群の x 座標を足し合わせていく
338         sum_pointcloud.y = sum_pointcloud.y + inputPointCloud->points[i].y; //点群の y 座標を足し合わせていく
339         sum_pointcloud.z = sum_pointcloud.z + inputPointCloud->points[i].z; //点群の z 座標を足し合わせていく
340         //cout << i << " : " << "sum_x => " << sum_pointcloud.x << ", sum_y => " << sum_pointcloud.y << ",
    sum_z => " << sum_pointcloud.z << endl;
341         //cout << sum_pointcloud << endl; //確認用
342     }
343     //cout << "SUM => " << sum_pointcloud << endl; //合計の確認用
344     centroid_coordinate.x = sum_pointcloud.x / inputPointCloud->size(); //x 座標の平均（重心）
345     centroid_coordinate.y = sum_pointcloud.y / inputPointCloud->size(); //y 座標の平均（重心）
346     centroid_coordinate.z = sum_pointcloud.z / inputPointCloud->size(); //z 座標の平均（重心）
347     //cout << "Centroid" << centroid_coordinate << endl; //確認用
348
349     //平均座標（重心）をファイルに出力（確認用）
350     //fprintf_s(centroid, "%f %f %f\n",centroid_coordinate.x,centroid_coordinate.y,centroid_coordinate.z);
351
352     //ファイルを閉じる（核に尿）
353     //fclose(pointcloud);
354     //fclose(centroid);
355
356     return centroid_coordinate;
357 }
```

3.10.3.4 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::getExtractPlaneAndClustering (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*, bool *optimize*, int *maxIterations*, double *threshold*, bool *negative1*, bool *negative2*, double *tolerance*, int *minClusterSize*, int *maxClusterSize*)

平面検出とクラスタリング

メソッドPointCloudLibrary::extractPlane(). 平面を検出するメソッド

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 214 行目に定義があります。

参照元 main().

```
215 {
216     cout << "before\tExtract Plane\t\t=>\t" << inputPointCloud->size() << endl;
217
218     pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_plane(new pcl::PointCloud<pcl::PointXYZRGB>());
219     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>());
220     pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients);
221     pcl::PointIndices::Ptr inliers(new pcl::PointIndices);
222
223     //セグメンテーションオブジェクトの生成
224     pcl::SACSegmentation<pcl::PointXYZRGB> seg;
225
226     //オプション
227     seg.setOptimizeCoefficients(optimize);
228
229     //Mandatory
230     seg.setModelType(pcl::SACMODEL_PLANE);
231     seg.setMethodType(pcl::SAC_RANSAC);
232
233     //クラスタリング
234     seg.setMaxIterations(maxIterations); //Default->100
235     //
236
237     seg.setDistanceThreshold(threshold);
238
239     int i = 0, nr_points = (int)inputPointCloud->points.size();
```

```
240     while (inputPointCloud->points.size() > 0.3*nr_points)
241     {
242         seg.setInputCloud(inputPointCloud);
243         seg.segment(*inliers, *coefficients);
244         if (inliers->indices.size() == 0)
245         {
246             cout << "Could not estimate a planar model for the given dataset." << endl;
247             break;
248         }
249         pcl::ExtractIndices<pcl::PointXYZRGB> extract;
250         extract.setInputCloud(inputPointCloud);
251         extract.setIndices(inliers);
252         extract.setNegative(negative1); //true:平面以外を残す. false:平面を残す
253 
254         extract.filter(*cloud_plane);
255         //cout << "PointCloud representing the planar component: " << cloud_plane->points.size() << endl;
    //平面のサイズ
256 
257         extract.setNegative(negative2);
258         extract.filter(*filtered);
259         pcl::copyPointCloud(*filtered, *inputPointCloud);
260     }
261 
262     pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZRGB>);
263     tree->setInputCloud(inputPointCloud);
264 
265     std::vector<pcl::PointIndices> cluster_indices;
266     pcl::EuclideanClusterExtraction<pcl::PointXYZRGB> ec;
267     ec.setClusterTolerance(tolerance); //単位 [m]
268     ec.setMinClusterSize(minClusterSize); //最小クラスタの値
269     ec.setMaxClusterSize(maxClusterSize); //最大クラスタの値
270     ec.setSearchMethod(tree); //検索手法
271     ec.setInputCloud(inputPointCloud); //点群を入力
272     ec.extract(cluster_indices); //クラスタ情報を出力
273 
274     int j = 0; //クラスタのカウント
275     pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_cluster(new pcl::PointCloud<pcl::PointXYZRGB>); //
    クラスタに色付後の点群用
276     pcl::PointCloud<pcl::PointXYZRGB>::Ptr max_cluster(new pcl::PointCloud<pcl::PointXYZRGB>); //
    最大のクラスタを探す (c76)
277     max_cluster = cloud_cluster; //最初のクラスタを最大クラスタとする (c76)
278 
279     float colors[6][3] = { { 255, 0, 0 }, { 0, 255, 0 }, { 0, 0, 255 }, { 255, 255, 0 }, { 0, 255, 255 }, {
    255, 0, 255 } }; //クラスタに色を付ける用
280     for (std::vector<pcl::PointIndices>::const_iterator it = cluster_indices.begin(); it != cluster_indices
    .end(); ++it) //クラスタを 1 塊ごとに出力
281     {
282         for (std::vector<int>::const_iterator pit = it->indices.begin(); pit != it->indices.end(); ++pit){
283             inputPointCloud->points[*pit].r = colors[j % 6][0];
284             inputPointCloud->points[*pit].g = colors[j % 6][1];
285             inputPointCloud->points[*pit].b = colors[j % 6][2];
286             cloud_cluster->points.push_back(inputPointCloud->points[*pit]);
287         }
288 
289         //最大のクラスタを探す. 元の mac_cluster より新しいクラスタの方が大きければ新しいクラスタを max_cluster とする (c76)
290         if (max_cluster->size() < cloud_cluster->size()){
291             pcl::copyPointCloud(*cloud_cluster, *max_cluster);
292         }
293 
294         //cloud_cluster->width = cloud_cluster->points.size();
295         //cloud_cluster->height = 1;
296         cloud_cluster->is_dense = true;
297         cout << "Cluster " << j << "\t\t\t=>\t" << cloud_cluster->size() << endl;
298 
299         j++;
300     }
301 
302     //cout << "width => " << max_cluster->width << " height => " << max_cluster->height << endl;
303 
304     //cout << "cloud_cluster => " << cloud_cluster->size() << ", max_cluster => " << max_cluster->size() <<
    endl;
305     //pcl::copyPointCloud(*cloud_cluster, *filtered); //カラーリングしたクラスタ全てを出力
306     pcl::copyPointCloud(*max_cluster, *filtered); //最大のクラスタのみ出力 (c76)
307     cout << "after\tExtract Plane\t\t=>\t" << filtered->size() << endl;
308     return filtered;
309 }
```

3.10.3.5 pcl::PointCloud< pcl::Normal >::Ptr PointCloudLibrary::getSurfaceNormals (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

法線を計算する

メソッドPointCloudLibrary::getSurfaceNormals(). 法線を計算する

引数

pcl::PointCloud<pcl::PointXYZRGB>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::Normal>::Ptr cloud_normals

PointCloudLibrary.cpp の 364 行目に定義があります。

```
365 {
366     pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> ne;
367     ne.setInputCloud(inputPointCloud); //入力された点群の法線を計算する
368     pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZRGB>());
369     ne.setSearchMethod(tree);
370     pcl::PointCloud<pcl::Normal>::Ptr cloud_normals(new pcl::PointCloud<pcl::Normal>);
371     ne.setRadiusSearch(0.005);
372     ne.compute(*cloud_normals);
373
374     cout << *cloud_normals << endl;
375     return cloud_normals;
376 }
```

3.10.3.6 void PointCloudLibrary::initializePCLVisualizer (string *pclvisualizer_name*)

PointCloudLibrary.cpp の 53 行目に定義があります。

参照先 visualizer.

参照元 main().

```
54 {
55     visualizer = new pcl::visualization::PCLVisualizer(pclvisualizer_name); //PCLVisualizer のウインドウ名
56     visualizer->setBackgroundColor(0, 0, 0); //PCLVisualizer の背景色
57     //visualizer->setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 1, "show
       cloud");
58     visualizer->addCoordinateSystem(1.0);
59     visualizer->initCameraParameters();
60     return;
61 }
```

3.10.3.7 void PointCloudLibrary::initializePointCloudViewer (string *cloudviewer_name*)

ポイントクラウドビューアーを初期化

メソッドPointCloudLibrary::initializePointCloudViewer(). ポイントクラウドビューアーを初期化するメソッド (c57)

引数

string	cloudviewer_name

PointCloudLibrary.cpp の 47 行目に定義があります。

参照先 viewer.

```
48 {
49     viewer = new pcl::visualization::CloudViewer(cloudviewer_name);
50     return;
51 }
```

3.10.3.8 void PointCloudLibrary::loadPLY (char * *ply_name*)

メソッドPointCloudLibrary::loadPLY(). ply ファイルを読み込む

引数

char∗	ply_name

PointCloudLibrary.cpp の 67 行目に定義があります。

参照先 model.

```
68 {
69     pcl::io::loadPLYFile(ply_name, *model); //ply ファイルを読み込み model に格納する
70     cout << "load " << ply_name << ". " << endl;
71     return;
72 }
```

3.10.3.9 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::passThroughFilter (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*, char ∗ *axis*, float *min*, float *max*)

パススルーフィルタ．z の値の距離に応じてカット可能

メソッドPointCloudLibrary::passThroughFilter(). パススルーフィルタ

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 103 行目に定義があります。

参照元 main().

```
104 {
105     cout << "before\tPassThroughFilter\t=>\t" << inputPointCloud->size() << endl;
106
107     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>()); //
      フィルター後用のポイントクラウド
108     pcl::PassThrough<pcl::PointXYZRGB> pass;
109     pass.setInputCloud(inputPointCloud);
110     pass.setFilterFieldName(axis);
111     pass.setFilterLimitsNegative(false); //setFilterLimits(float min, float max) で指定した範囲以外を削除 (c69)
112     pass.setFilterLimits(min, max);
113     pass.filter(*filtered);
114
115     cout << "after\tPassThroughFilter\t=>\t" << filtered->size() << endl;
116     return filtered;
117 }
```

3.10.3.10 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::radiusOutlierRemoval (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

メソッドPointCloudLibrary::radiusOutlierRemoval(). 外れ値を除去するメソッド (c60)

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 146 行目に定義があります。

```
147 {
148     cout << "before\tRadiusOutlierRemoval\t=>\t" << inputPointCloud->size() << endl;
149
150     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>()); //
      フィルタリング後用のポイントクラウドを宣言
151     pcl::RadiusOutlierRemoval<pcl::PointXYZRGB> ror;
152
153     ror.setInputCloud(inputPointCloud);
154     ror.setRadiusSearch(0.8);
155     ror.setMinNeighborsInRadius(2);
156     ror.filter(*filtered);
157
158     cout << "after\tRadiusOutlierRemoval\t=>\t" << filtered->size() << endl;
159     return filtered;
160 }
```

3.10.3.11 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::removeOutlier (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

メソッドPointCloudLibrary::removeOutlier(). 外れ値を除去するメソッド (c59)

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 124 行目に定義があります。

参照元 main().

```
125 {
126     cout << "before\tRemove Outlier\t\t=>\t" << inputPointCloud->size() << endl;
127
128     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>()); //
      フィルタリング後用のポイントクラウドを宣言
129     pcl::StatisticalOutlierRemoval<pcl::PointXYZRGB> fl;
130
131     fl.setInputCloud(inputPointCloud);
132     fl.setMeanK(10);
133     fl.setStddevMulThresh(0.1);
134     fl.filter(*filtered);
135     fl.setNegative(true);
136
137     cout << "after\tRemove Outlier\t\t=>\t" << filtered->size() << endl;
138     return filtered;
139 }
```

3.10.3.12 pcl::PointCloud< pcl::PointXYZRGB >::Ptr PointCloudLibrary::smoothingUsingMovingLeastSquare (pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*, bool *compute_normals*, bool *polynomial_fit*, double *radius*)

スムージング

メソッドPointCloudLibrary::smoothingUsingMovingLeastSquare(). スムージングを行うメソッド (c60)

引数

pcl::PointCloud<pcl::PointXYZ>::Ptr	&inputPointCloud

戻り値

pcl::PointCloud<pcl::PointXYZ>::Ptr filtered

PointCloudLibrary.cpp の 189 行目に定義があります。

参照元 main().

```
190 {
191     cout << "before\tMLS\t\t\t=>\t" << inputPointCloud->size() << endl;
192
193     pcl::PointCloud<pcl::PointXYZRGB>::Ptr filtered(new pcl::PointCloud<pcl::PointXYZRGB>()); //
      フィルタリング処理後用のポイントクラウド
194     pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZRGB>()); //
      KdTree の作成
195     pcl::MovingLeastSquares<pcl::PointXYZRGB, pcl::PointXYZRGB> mls; //スムージング処理
196
197     mls.setComputeNormals(compute_normals); //法線の計算
198     //各パラメータの設定
199     mls.setInputCloud(inputPointCloud);
200     mls.setPolynomialFit(polynomial_fit);
201     mls.setSearchMethod(tree);
202     mls.setSearchRadius(radius);
203     mls.process(*filtered); // 出力
204
205     cout << "after\tMLS\t\t\t=>\t" << filtered->size() << endl;
206     return filtered;
207 }
```

3.10.4 メンバ詳解

3.10.4.1 Point3d PointCloudLibrary::centroid

平均座標

PointCloudLibrary.hpp の 44 行目に定義があります。

参照元 main().

3.10.4.2 bool PointCloudLibrary::downsampling_flag

PointCloudLibrary.hpp の 57 行目に定義があります。

参照元 flagChecker(), main(), PointCloudLibrary().

3.10.4.3 bool PointCloudLibrary::extractplane_flag

PointCloudLibrary.hpp の 60 行目に定義があります。

参照元 flagChecker(), main(), PointCloudLibrary().

3.10.4.4 bool PointCloudLibrary::mls_flag

PointCloudLibrary.hpp の 59 行目に定義があります。

参照元 flagChecker(), main(), PointCloudLibrary().

3.10.4.5 pcl::PointCloud<pcl::PointXYZRGB>::Ptr PointCloudLibrary::model

PointCloudLibrary.hpp の 32 行目に定義があります。
参照元 loadPLY().

3.10.4.6 bool PointCloudLibrary::passthrough_flag

PointCloudLibrary.hpp の 56 行目に定義があります。
参照元 flagChecker(), main(), PointCloudLibrary().

3.10.4.7 bool PointCloudLibrary::statisticaloutlierremoval_flag

PointCloudLibrary.hpp の 58 行目に定義があります。
参照元 flagChecker(), main(), PointCloudLibrary().

3.10.4.8 pcl::visualization::CloudViewer* PointCloudLibrary::viewer

PointCloudLibrary.hpp の 50 行目に定義があります。
参照元 initializePointCloudViewer().

3.10.4.9 pcl::visualization::PCLVisualizer* PointCloudLibrary::visualizer

PointCloudLibrary.hpp の 53 行目に定義があります。
参照元 initializePCLVisualizer(), main().
このクラス詳解は次のファイルから抽出されました:

- **PointCloudLibrary.hpp**
- **PointCloudLibrary.cpp**

3.11 System クラス

`#include <System.hpp>`

公開メンバ関数

- **System** ()
  コンストラクタ
- ~**System** ()
  デストラクタ
- void **countdownTimer** (int time_sec)
  引数の時間 *[sec]* に応じてカウントダウンを開始する *(c75)*
- void **startMessage** ()
  プログラム開始時のメッセージを表示 *(c26)*
- void **endMessage** (int cNum)
  プログラム終了時のメッセージを表示 *(c38)*
- void **endMessage** ()
  プログラム終了時のメッセージを表示 *(c63)*
- void **startTimer** ()

タイマーを開始 *(c65)*

- void **endTimer** ()

  タイマーを終了 *(c65)*

- double **getProcessTimeinMiliseconds** ()

  計測した時間をミリ秒単位で取得.*startTimer()* と *endTimer()* が実行されていることが前提 *(c65)*

- double **getFrameRate** ()

  フレームレートを取得.*startTimer()* と *endTimer()* が実行されていることが前提 *(c65)*

- void **checkDirectory** (const char *check_dirname)

  引数に与えたファイルやディレクトリが存在するかチェックし，無ければ作成する *(c81)*

- void **makeDirectory** ()

  System::makeDirectory() (p. *43*). ディレクトリを作成

- void **removeDirectory** ()

  取得したデータが不要だった場合ディレクトリを削除する

- int **alternatives** ()

  数字の入力をチェックする

- void **openDirectory** ()

  ディレクトリを開く *(c38)*

- void **outputAllData** (const string *outputDataName, **outputData** *__outputData__, int countDataNum)

  データをファイルに書き出すメソッド *(c41)*

- VideoWriter **outputVideo** (const string *outputVideoName)

  動画を出力する

- void **saveDataEveryEnterKey** (Mat &current_image, Mat &bin_image, **DoF6d** dof6, pcl::PointCloud< pcl::-PointXYZRGB >::Ptr &inputPointCloud)

  メソッド*System::saveDataEveryEnterKey()*. 連続で計測している際に *p* キーを入力するとその時点のデータを新しいディレクトリに保存する.

- void **saveDataContinuously** (**DoF6d** centroid)

  メソッド*System::saveDataContinuously()*. *p* キーが入力されたら，平均座標を出力 *(c82) @param DoF6d centroid*

公開変数類

- bool **save_flag**

  6DoF 情報を出力するフラグ

3.11.1 詳解

System.hpp の 17 行目に定義があります。

3.11.2 構築子と解体子

3.11.2.1 System::System ( )

コンストラクタ

System::System() (p. 39). コンストラクタ

System.cpp の 16 行目に定義があります。

参照先 save_flag.

```
17 {
18     FlagStartTimer = false; //スタート用のタイマーが実行されたかのフラグを初期化 (c65)
19     FlagEndTimer = false; //終了用のタイマーが実行されたかのフラグを初期化 (c65)
20     time = 0.0; //時間計測用の変数を初期化 (c65)
21     //save_count = 0; //保存用カウント用の変数を初期化 (c82)
22     save_flag = false;
23 }
```

3.11.2.2 System::~System ( )

デストラクタ

System::~System() (p. 40). デストラクタ

System.cpp の 28 行目に定義があります。

```
29 {
30     //デストラクタの処理はなし
31 }
```

3.11.3 関数詳解

3.11.3.1 int System::alternatives ( )

数字の入力をチェックする

System::alternatives() (p. 40).Yes/No の 2 択のチェック (c21)

戻り値

checkNum

<0 か 1 をチェックするための変数 (c27). このメソッドのみで有効な変数 (c30)

System.cpp の 261 行目に定義があります。

参照元 main().

```
262 {
263     char checkNum;
264 
265     cout << "1. Yes" << endl;
266     cout << "0. No" << endl;
267     cin >> checkNum;
268     cout << "\n";
269 
270     while (1)
271     {
272         if (checkNum == '0'){
273             return (atoi(&checkNum));
274         }
275         else if (checkNum == '1'){
276             return (atoi(&checkNum));
277         }
278         else{
279             cout << "Please Input 1 or 0." << endl;
280             cout << "1. Yes" << endl;
281             cout << "0. No" << endl;
282             cin >> checkNum;
283         }
284     }
285 }
```

3.11.3.2 void System::checkDirectory (const char * *check_dirname*)

引数に与えたファイルやディレクトリが存在するかチェックし，無ければ作成する (c81)

System::checkDirectory() (p. 40). ファイルやディレクトリがあるかチェックし，無ければ作成する (c81)

引数

string	checkdir_name

System.cpp の 213 行目に定義があります。

参照元 main().

```
214 {
215     struct stat st;
216     int re = stat(check_dirname, &st);
217     if (re != 0){
218         _mkdir(check_dirname);
219         cout << "Make " << check_dirname << ". " << endl;
220     }
221     return;
222 }
```

3.11.3.3 void System::countdownTimer (int *time_sec*)

引数の時間 [sec] に応じてカウントダウンを開始する (c75)

メソッド countdownTimer(). タイマーによるカウントダウン

引数

int	time_sec

System.cpp の 88 行目に定義があります。

参照元 main().

```
89 {
90     string playfile_name;
91     while (time_sec > 0){
92         cout << time_sec << " seconds" << "\r";
93         switch (time_sec)
94         {
95         case 1:
96             PlaySound(TEXT("sourcedata/count1.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
97             break;
98         case 2:
99             PlaySound(TEXT("sourcedata/count2.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
100             break;
101         case 3:
102             PlaySound(TEXT("sourcedata/count3.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
103             break;
104         case 4:
105             PlaySound(TEXT("sourcedata/count4.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
106             break;
107         case 5:
108             PlaySound(TEXT("sourcedata/count5.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
109             break;
110         case 6:
111             PlaySound(TEXT("sourcedata/count6.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
112             break;
113         case 7:
114             PlaySound(TEXT("sourcedata/count7.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
115             break;
116         case 8:
117             PlaySound(TEXT("sourcedata/count8.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
118             break;
119         case 9:
120             PlaySound(TEXT("sourcedata/count9.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
121             break;
122         case 10:
123             PlaySound(TEXT("sourcedata/count10.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
124             break;
125         default: break;
126         }
127         Sleep(1000);
128         time_sec = time_sec - 1;
129     }
130     cout << "\n";
131     return;
132 }
```

3.11.3.4 void System::endMessage (int *cNum*)

プログラム終了時のメッセージを表示 (c38)

System::endMessage() (p. 42). プログラム終了時のメッセージ

System.cpp の 60 行目に定義があります。

参照先 directoryName, openDirectory().

参照元 main().

```
61 {
62     cout << "=================================================================" << endl;
63     cout << "Closing the Program...." << endl;
64     if (cNum == 1){
65         cout << "Data Has Been Output to \"data/" << directoryName << "\"." << endl;
66         openDirectory();
67     }
68     cout << "=================================================================" << endl;
69
70     return;
71 }
```

3.11.3.5 void System::endMessage ( )

プログラム終了時のメッセージを表示 (c63)

System::endMessage() (p. 42). プログラム終了時のメッセージ

System.cpp の 76 行目に定義があります。

```
77 {
78     cout << "=================================================================" << endl;
79     cout << "Closing the Program...." << endl;
80     cout << "=================================================================" << endl;
81     return;
82 }
```

3.11.3.6 void System::endTimer ( )

タイマーを終了 (c65)

メソッド endTimer(). タイマーを終了する

System.cpp の 148 行目に定義があります。

参照元 main().

```
149 {
150     if (FlagStartTimer == true){
151         end = getTickCount();
152         time = (end - start) * f;
153         FlagEndTimer = true;
154     }
155     else{
156         cerr << "Before you use endTimer() method, please run the System::startTimer()." << endl;
157         exit(-1);
158     }
159     return;
160 }
```

3.11.3.7 double System::getFrameRate ( )

フレームレートを取得.startTimer() と endTimer() が実行されていることが前提 (c65)

メソッド getFrameRate(). フレームレートを取得する (c65)

戻り値

double time

System.cpp の 189 行目に定義があります。

参照元 main().

```
190 {
191     if (FlagStartTimer == true && FlagEndTimer == true){
192         fps = 1000.0 / time;
193         return fps;
194     }
195     else if (FlagStartTimer == false && FlagEndTimer == true){
196         cerr << "Before you use getFrameRate() method, please run the System::startTimer()." << endl;
197         exit(-1);
198     }
199     else if (FlagStartTimer == true && FlagEndTimer == false){
200         cerr << "Before you use getFrameRate() method, please run the System::endTimer()." << endl;
201         exit(-1);
202     }
203     else{
204         cerr << "Before you use getFrameRate() method, please run the System::startTimer() and
       System::endTimer()." << endl;
205         exit(-1);
206     }
207 }
```

3.11.3.8 double System::getProcessTimeinMiliseconds ( )

計測した時間をミリ秒単位で取得.startTimer() と endTimer() が実行されていることが前提 (c65)

メソッド getTime(). 計測した時間を取得する (c65)

戻り値

double time

System.cpp の 166 行目に定義があります。

参照元 main().

```
167 {
168     if (FlagStartTimer == true && FlagEndTimer == true){
169         return time;
170     }
171     else if (FlagStartTimer == false && FlagEndTimer == true){
172         cerr << "Before you use getProcessTimeinMiliseconds() method, please run the System::startTimer()."
      << endl;
173         exit(-1);
174     }
175     else if (FlagStartTimer==true && FlagEndTimer == false){
176         cerr << "Before you use getProcessTimeinMiliseconds() method, please run the System::endTimer()." <
      < endl;
177         exit(-1);
178     }
179     else{
180         cerr << "Before you use getProcessTimeinMiliseconds() method, please run the System::startTimer()
      and System::endTimer()." << endl;
181         exit(-1);
182     }
183 }
```

3.11.3.9 void System::makeDirectory ( )

System::makeDirectory() (p. 43). ディレクトリを作成

System.cpp の 227 行目に定義があります。

参照先 directoryName, NOC.

参照元 main().

```
228 {
229     //フォルダ名を区別するために時刻を取得しておく (c4)
230     SYSTEMTIME st;
231     char savedirpath[NOC];
232
233     GetLocalTime(&st);
234     sprintf_s(directoryName, "[%4d%02d%02d]%02d%02d%02d", st.wYear, st.wMonth, st.wDay, st.wHour, st.
      wMinute, st.wSecond);
```

```
235     sprintf_s(savedirpath, "data/%s", directoryName);
236     _mkdir(savedirpath);
237
238     return;
239 }
```

3.11.3.10 void System::openDirectory ( )

ディレクトリを開く (c38)

System::openDireectory(). 出力したディレクトリを開く (c39)

System.cpp の 290 行目に定義があります。

参照先 directoryName, NOC.

参照元 endMessage().

```
291 {
292     cout << "Opening This Directory." << endl;
293     char openDirectoryCommand[NOC]; //自動で開くウインドウのパス
294     sprintf_s(openDirectoryCommand, "explorer \"data\\%s\"", directoryName); //ディレクトリを開くコマンド
295     system(openDirectoryCommand); //ディレクトリを開くコマンドを実行
296
297     return;
298 }
```

3.11.3.11 void System::outputAllData (const string * *outputDataName*, outputData * *outputData*, int *countDataNum*)

データをファイルに書き出すメソッド (c41)

System::outputData(). データをファイルに書き出すメソッド (c41)

引数

outputDataName,***outputData*** *(p. 26)*,*countDataNum*	

< 抽出した座標の距離 (c7)

< ファイルポインタの初期化 (c7)

< データファイル出力の際のパス (c37)

System.cpp の 304 行目に定義があります。

参照先 directoryName, NOC.

```
305 {
306     //ファイルポインタ
307     FILE *extractedCoordinate;
308     extractedCoordinate = NULL;
309
310     //(X,Y,Z) データ格納用のファイル
311     char outputDataPath[NOC];
312
313     sprintf_s(outputDataPath, "%s/%s", directoryName, outputDataName); //(c7)
314     fopen_s(&extractedCoordinate, outputDataPath, "w"); //(c7)
315     if (outputDataPath == NULL){ //ファイルオープンエラー処理 (c40)
316         cerr << outputDataPath << " is Not Opened.";
317         exit(1);
318     }
319
320     //ファイルに出力する処理 (c42)
321     for (int i = 2; i < countDataNum - 15; i++){ //最初と最後のいくつかのデータをファイルに出力しない (c41)
322         fprintf_s(extractedCoordinate, "%f %f %f %f\n", outputData[i].totalTime, outputData[i].x,
    outputData[i].y, outputData[i].z);
323     }
324     fclose(extractedCoordinate); //(c8)
```

```
325
326     return;
327 }
```

3.11.3.12 VideoWriter System::outputVideo (const string * *outputVideoName*)

動画を出力する

System::outputVideo() (p. 45). 動作確認用に動画を出力するメソッド

引数

cameraParam-File	

戻り値

VideoWriter writer

< 動画出力時のパス (c38)

System.cpp の 334 行目に定義があります。

参照先 directoryName, NOC.

```
335 {
336     //動画を出力 (c40)
337     char outputVideoPath[NOC];
338     sprintf_s(outputVideoPath, "%s/%s", directoryName, outputVideoName); //(c38)
339     //VideoWriter writer(outputVideoPath, /*CV_FOURCC('D','I','B','
       ')*/-1/*CV_FOURCC('X','V','I','D')*//*CV_FOURCC('P','M','I','1')*/, 20, /*Size(WIDTH, HEIGHT)*/Size(640, 480), true); 
動画に出力．録画が必要なときはコメントアウト (c35)
340     VideoWriter writer(outputVideoPath, /*CV_FOURCC('D','I','B',' ')*/-1/*CV_FOURCC('X','V','I','D')*//*
      CV_FOURCC('P','M','I','1')*/, 20, /*Size(WIDTH, HEIGHT)*/Size(640, 480), true); //動画に出力．録画が必要なと
きはコメントアウト (c35)
341     if (!writer.isOpened()){ //オープンエラー処理 (c40)
342         cerr << outputVideoPath << " is Not Opened." << endl;
343     }
344     return (writer);
345 }
```

3.11.3.13 void System::removeDirectory ( )

取得したデータが不要だった場合ディレクトリを削除する

System::removeDirectory() (p. 45). ディレクトリを削除 (c21)

System.cpp の 244 行目に定義があります。

参照先 directoryName, NOC.

参照元 main().

```
245 {
246     char rmdirCommand[NOC]; //ディレクトリを削除するコマンド (c21)．変数名を変更&このメソッドのみで有効な変数 (c30)
247
248     //ディレクトリを削除する
249     cout << directoryName << endl;
250     sprintf_s(rmdirCommand, "rmdir /s /q \"data\\%s\"", directoryName); //
      ディレクトリの削除コマンドを書く．パスは [\"\"] で囲み，階層があるときは [\\] で書く
251     system(rmdirCommand);
252     cout << "Not Save.\n" << endl;
253
254     return;
255 }
```

3.11.3.14 void System::saveDataContinuously (DoF6d *centroid*)

メソッドSystem::saveDataContinuously(). p キーが入力されたら，平均座標を出力 (c82) @param DoF6d centroid

System.cpp の 409 行目に定義があります。

参照先 directoryName, NOC, DoF::x, DoF::y, DoF::z.

参照元 main().

```
410 {
411     //保存用のファイル作成
412     FILE *ev3route;
413     char filepath_ev3route[NOC];
414     sprintf_s(filepath_ev3route, "data/%s/ev3route.dat", directoryName);
415     fopen_s(&ev3route, filepath_ev3route, "a"); //ファイルオープン
416     fprintf_s(ev3route, "%f %f %f\n", centroid.x, centroid.y, centroid.z); //データをファイルに書き込む
417
418     fclose(ev3route);
419     return;
420 }
```

3.11.3.15 void System::saveDataEveryEnterKey (Mat & *current_image*, Mat & *bin_image*, DoF6d *dof6*, pcl::PointCloud< pcl::PointXYZRGB >::Ptr & *inputPointCloud*)

メソッドSystem::saveDataEveryEnterKey(). 連続で計測している際に p キーを入力するとその時点のデータを新しいディレクトリに保存する.

引数

cv::Mat&	current_image, cv::Mat& bin_image, DoF6d dof6, pcl::PointCloud<pcl::PointXYZRGB>::Ptr &inputPointCloud

System.cpp の 351 行目に定義があります。

参照先 directoryName, NOC, DoF::pitch, DoF::roll, save_count, save_flag, DoF::x, DoF::y, DoF::yaw, DoF::z.

参照元 main().

```
352 {
353     //その都度保存するためのディレクトリを作成 (c82)
354     char filepath_output[NOC];
355     sprintf_s(filepath_output, "data/%s/%d", directoryName, save_count);
356     _mkdir(filepath_output);
357
358     //現在の画像を保存
359     char filepath_currentimage[NOC];
360     sprintf_s(filepath_currentimage, "data/%s/%d/current_image-%02d.jpg",
      directoryName, save_count, save_count);
361     imwrite(filepath_currentimage, current_image);
362
363     //差分を計算した二値画像を保存
364     char filepath_binimage[NOC];
365     sprintf_s(filepath_binimage, "data/%s/%d/background_image-%02d.jpg",
      directoryName, save_count, save_count);
366     imwrite(filepath_binimage, bin_image);
367
368     //点群情報を保存
369     FILE *point_fp; //最終 1 フレーム分. gnuplot で表示するために点群をファイルに出力する用
370     char filepath_point[NOC];
371     sprintf_s(filepath_point, "data/%s/%d/point-%02d.dat", directoryName,
      save_count, save_count);
372     fopen_s(&point_fp, filepath_point, "w"); //
373     for (int i = 1; i < inputPointCloud->size(); i++){
374         fprintf_s(point_fp, "%f %f %f\n", inputPointCloud->points[i].x, inputPointCloud->points[i].y,
      inputPointCloud->points[i].z); //ファイルに出力
375     }
376     fclose(point_fp);
377
378     //6DoF 情報を保存
379     FILE *dof6_fp; //最終 1 フレーム分. gnuplot で表示するために平均座標 (重心) をファイルに出力する用
380     char filepath_dof6[NOC];
381     sprintf_s(filepath_dof6, "data/%s/%d/dof6-%02d.dat", directoryName,
      save_count, save_count);
382     fopen_s(&dof6_fp, filepath_dof6, "w");
383     fprintf_s(dof6_fp, "%f %f %f %f %f %f %d\n", dof6.x, dof6.y, dof6.z, dof6.
```

```
      yaw, dof6.roll, dof6.pitch, inputPointCloud->size());
384     fclose(dof6_fp);
385
386     //6DoF 情報を続けて保存する
387     FILE *dof6con_fp;
388     char filepath_dof6con[NOC];
389     sprintf_s(filepath_dof6con, "data/%s/dof6con.csv", directoryName);
390     fopen_s(&dof6con_fp, filepath_dof6con, "a");
391     if (save_flag == false){
392         fprintf(dof6_fp, "x,y,z,Yaw,Roll,Pitch,Data Size\n");
393     }
394     fprintf_s(dof6_fp, "%f,%f,%f,%f,%f,%f,%d\n", dof6.x, dof6.y, dof6.z, dof6.
      yaw, dof6.roll, dof6.pitch, inputPointCloud->size());
395     fclose(dof6con_fp);
396
397     //PointCloud を保存する
398     char filepath_pointcloud[NOC];
399     sprintf_s(filepath_pointcloud, "data/%s/%d/pointcloud-%02d.ply",
      directoryName, save_count, save_count);
400     pcl::io::savePLYFileASCII(filepath_pointcloud, *inputPointCloud);
401
402     return;
403 }
```

3.11.3.16 void System::startMessage ( )

プログラム開始時のメッセージを表示 (c26)

System::startMessage() (p. 47). プログラム起動時のメッセージ

System.cpp の 36 行目に定義があります。

参照元 main().

```
37 {
38     cout << "=================================================================" << endl;
39     cout << " Starting the Program...." << endl;
40     //cout << " Please Enclose the Object You Want to Track." << endl;
41     cout << " If You Enter a 'q' Key, the Program Terminates." << endl;
42     cout << " If You Enter a 'r' Key, the Program Restart." << endl;
43     cout << " Each Time You Enter a 'p' Key, Then Save The Data." << endl;
44     //cout << " To Initialize Tracking, Re-Select the Object with Mouse." << endl;
45     cout << "\n";
46     cout << " Switching of Point Cloud Processing." << endl;
47     cout << "  Pass Through Filter \t\t -> \t Enter 'x' Key." << endl;
48     cout << "  Downsampling \t\t\t -> \t Enter 'c' Key." << endl;
49     cout << "  Remove Outlier \t\t -> \t Enter 'v' Key." << endl;
50     cout << "  Moving Least Square \t\t -> \t Enter 'b' Key." << endl;
51     cout << "  Extract Plane & Clustering \t -> \t Enter 'n' Key." << endl;
52     cout << "=================================================================" << endl;
53
54     return;
55 }
```

3.11.3.17 void System::startTimer ( )

タイマーを開始 (c65)

メソッド startTimer(). タイマーを開始する

System.cpp の 137 行目に定義があります。

参照元 main().

```
138 {
139     f = 1000.0 / getTickFrequency();
140     start = getTickCount(); //スタート
141     FlagStartTimer = true; //スタート用のタイマーが実行されたのでフラグを true に
142     return;
143 }
```

3.11.4 メンバ詳解

3.11.4.1 bool System::save_flag

6DoF 情報を出力するフラグ

System.hpp の 62 行目に定義があります。

参照元 main(), saveDataEveryEnterKey(), System().

このクラス詳解は次のファイルから抽出されました:

- **System.hpp**
- **System.cpp**

Chapter 4

ファイル詳解

4.1 Drawing.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "Drawing.hpp"
```

4.2 Drawing.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **Drawing**

  経路描画用のクラス

4.3 EV3Control.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "EV3Control.hpp"
```

4.4 EV3Control.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **EV3Control**

  EV3 を制御するためのクラス

4.5 ImageProcessing.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "ImageProcessing.hpp"
```

4.6 ImageProcessing.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **ImageProcessing**

  画像処理用のクラス

4.7 Kinect.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "Kinect.hpp"
```

4.8 Kinect.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **Kinect**

  Kinect 操作用のクラス

マクロ定義

- #define **ERROR_CHECK**(ret)

  エラーチェック

変数

- const NUI_IMAGE_RESOLUTION **CAMERA_RESOLUTION** = NUI_IMAGE_RESOLUTION_640x480

4.8.1 マクロ定義詳解

4.8.1.1 #define ERROR_CHECK(*ret*)

値:

```
if (ret != S_OK){ \
stringstream ss; \
ss << "faild" #ret " " << hex << ret << endl; \
throw runtime_error(ss.str().c_str()); \
}
```

エラーチェック

Kinect.hpp の 17 行目に定義があります。

参照元 Kinect::createInstance(), Kinect::drawRGBImage(), Kinect::getPointCloud(), Kinect::initialize().

4.8.2 変数詳解

4.8.2.1 const NUI_IMAGE_RESOLUTION CAMERA_RESOLUTION = NUI_IMAGE_RESOLUTION_640x480

Kinect の解像度の設定

Kinect.hpp の 25 行目に定義があります。

参照元 Kinect::getPointCloud(), Kinect::initialize().

4.9 LeastSquareMethod.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "LeastSquareMethod.hpp"
```

4.10 LeastSquareMethod.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **LeastSquareMethod**
  最小二乗法を行うクラス

4.11 main.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "Kinect.hpp"
#include "ImageProcessing.hpp"
#include "Drawing.hpp"
#include "System.hpp"
#include "LeastSquareMethod.hpp"
#include "PointCloudLibrary.hpp"
#include "EV3Control.hpp"
```

関数

- void **onMouse** (int event, int x, int y, int flags, void *param)

マウス操作

- int **main** ()

  関数 *main()*

変数

- Mat **image**

  RGB 画像格納用の変数
- char **directoryName** [**NOC**]

  フォルダ名
- bool **selectObject** = false

  オブジェクト選択
- int **trackObject** = 0

  追跡するオブジェクト
- Point **origin**

  オリジナルの座標
- Rect **selection**

  選択
- int **save_count** = 0

4.11.1 関数詳解

4.11.1.1 int main ( )

関数 main()

引数

なし	

戻り値

なし

＜システム的なメソッドをまとめているクラス

<drawing クラスのインスタンスを生成

＜最小二乗法を行うクラスのインスタンスを生成 (c49)

<EV3 を制御する用のクラスを作成 (c80)

<EV3 の軌道を保存するためのフラグ (c82)

＜背景画像 (c75)

pointcloudlibrary.viewer->wasStopped() &&

main.cpp の 39 行目に定義があります。

参照先 System::alternatives(), LeastSquareMethod::calcYawRollPitch(), PointCloudLibrary::centroid, System::checkDirectory(), ImageProcessing::closing_times, System::countdownTimer(), PointCloudLibrary::downsampling_flag, PointCloudLibrary::downSamplingUsingVoxelGridFilter(), Kinect::drawRGBImage(), System::endMessage(), System::endTimer(), EV3Control::ev3_6dof, PointCloudLibrary::extractplane_flag, PointCloudLibrary::flagChecker(), ImageProcessing::getBackgroundSubstractionBinImage(), PointCloudLibrary::getCentroidCoordinate3d(), LeastSquareMethod::getCoefficient(), PointCloudLibrary::getExtractPlaneAndClustering(), System::getFrameRate(), Kinect::getPointCloud(), System::getProcessTimeinMiliseconds(), ImageProcessing::getUndistortionImage(), Drawing::gnuplotScriptEV3Route(), Drawing::gnuplotScriptEV3Unit(), image, Kinect::initialize(), PointCloudLibrary::initializePCLVisualizer(), Kinect::key, ImageProcessing::loadInternalCameraParameter(), System::makeDirectory(), PointCloudLibrary::mls_flag, ImageProcessing::neighborhood, PointCloudLibrary::passthrough_flag, PointCloudLibrary::passThroughFilter(), System::removeDirectory(), PointCloudLibrary::removeOutlier(),

save_count, System::save_flag, System::saveDataContinuously(), System::saveDataEveryEnterKey(), EV3Control::set6DoFEV3(), ImageProcessing::showImage(), PointCloudLibrary::smoothingUsingMovingLeastSquare(), System::startMessage(), System::startTimer(), PointCloudLibrary::statisticaloutlierremoval_flag, Kinect::streamEvent, ImageProcessing::th, PointCloudLibrary::visualizer (計 49 項目).

```
40 {
41     RETRY: //goto 文．再計測をやり直す場合
42     //インスタンスの生成
43     Kinect kinect; //Kinect クラスのインスタンスを生成
44     System sys;
45     ImageProcessing imgproc; //Imageprocessing クラスのインスタンスを生成
46     Drawing draw;
47     LeastSquareMethod lsm;
48     PointCloudLibrary pointcloudlibrary(/*false*/true, /*false*/true, /*false*/true, false, false/*true*/);
       //PointCloudLibrary クラスのインスタンスを生成 (c57)
49     EV3Control ev3control;
50
51     //変数の宣言
52     bool saveev3route_flag = false;
53
54     //画像関係の変数
55     Mat flip_image; //確認用に反転した画像
56     Mat current_image; //現在のフレームの画像 (c75)
57     Mat bin_image; //背景差分によって得られた二値画像 (c75)
58     Mat background_image;
59     Mat background_gray_image;
60
61     //ポイントクラウド関係の変数 (c57)
62     pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud; //処理を受け取る点群
63
64     //EV3 ユニットの平面の係数 (c78)
65     Eigen::Vector3f coefficient_plane; //平面の係数
66     AttitudeAngle3d attitude_angle; //姿勢角 (c78)
67
68     //CloudVisualizerm の初期設定 (c83)
69
70
71     //メインの処理
72     try{
73         sys.startMessage(); //プログラム開始時のメッセージを表示
74         //初期設定 (利用する場合はコメントを外す)
75         //pointcloudlibrary.loadPLY("EV3COLOR.ply"); //.ply ファイルの読み込み．動作しない
76         //VideoWriter writer; //動画保存用
77
78         //ウインドウ名とファイル名の定義
79         const string main_windowname = "Current Image"; //メインウインドウの名前をつけておく．(c31)
80         const string backgroundimage_windowname = "Background Image"; //背景画像を表示するためのウインドウ名
81         const string maskbinimage_windowname = "Mask Image"; //マスク画像を表示するためのウインドウ名
82         //const string outputVideoName = "video.avi"; //計測中の動画ファイル名 (c39)
83         const string cameraparameter_name = "sourcedata/cameraParam.xml"; //
       xml ファイル名の定義．カメラキャリブレーションによって得られたファイル名 (c54)
84         const char* basedirectory_name = "data"; //データ保存先のディレクトリ名
85         const string cloudviewer_windowname = "Cloud Viewer"; //クラウドビューアーの名前の定義 (c81)
86         const string pclvisualizer_windowname = "3D Viewer";
87         const string param_windowname = "OpenCV Parameter Setting Window"; //パラメータ調整用のウインドウ (c82)
88
89         kinect.initialize(); //Kinect の初期化
90         sys.checkDirectory(basedirectory_name); //base_directory が存在するか確認し，存在しなければ作成 (c81)
91         sys.makeDirectory(); //起動時刻をフォルダ名にしてフォルダを作成
92
93         //動画を保存するために利用する (c40)
94         //writer = sys.outputVideo(&outputVideoName); //動画を保存したいときはコメントをはずす．while 文内の
writer << image のコメントも
95
96         imgproc.loadInternalCameraParameter(cameraparameter_name); //キャリブレーションを行うためのパラメータを取得
(c79)
97
98         //背景用に一度撮影 (c75)
99         cout << "Take a Background Image in " << endl;
100         sys.countdownTimer(3);
101         system("cls");
102         DWORD ret = ::WaitForSingleObject(kinect.streamEvent, INFINITE); //フレーム更新をイベントとして待つ
103         ::ResetEvent(kinect.streamEvent); //イベントが発生したら次のイベントに備えてリセット
104         //Kinect から画像を取得し，背景画像とする
105         PlaySound(TEXT("sourcedata/shutter.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
106         background_image = kinect.drawRGBImage(image); //RGB カメラの処理
107         background_image = imgproc.getUndistortionImage(background_image); //キャリブレーション後の画像を今の背
景画像とする
108         imgproc.showImage(backgroundimage_windowname, background_image); //背景画像を表示する
109         cvtColor(background_image, background_gray_image, CV_BGR2GRAY); //グレースケールに変換
110
111         Sleep(1000);
112
113         //プログラム開始の通知
114         cout << "Process will Start in " << endl;
```

```
115         sys.countdownTimer(2);
116         PlaySound(TEXT("sourcedata/shutter.wav"), NULL, (SND_ASYNC | SND_FILENAME)); //音声ファイルを再生
117         system("cls"); //コンソール内の表示をリセット (c64)
118 
119         //pointcloudlibrary.initializePointCloudViewer(cloudviewer_windowname); //クラウドビューアーの初期化
120         pointcloudlibrary.initializePCLVisualizer(pclvisualizer_windowname);
121 
122         //namedWindow("閾値", 1);
123         while (!pointcloudlibrary.visualizer->wasStopped() && kinect.key != 'q' && !GetAsyncKeyState('Q')){
    //(c3). メインループ. 1 フレームごとの処理を繰り返し行う. (c63)CloudViewer が終了処理 ('q' キーを入力) したらプログラムが
終了する
124             //タイマー開始 (c65)
125             sys.startTimer();
126 
127             //setMouseCallback(mainwindow_name, onMouse, 0); //(c25). マウスコールバック関数をセット (c31)
128 
129             DWORD ret = ::WaitForSingleObject(kinect.streamEvent, INFINITE); //フレーム更新をイベントとして待つ
130             ::ResetEvent(kinect.streamEvent); //イベントが発生したら次のイベントに備えてリセット
131 
132             //Kinect から画像を取得し, 画像処理を行う
133             current_image = kinect.drawRGBImage(image); //RGB カメラの処理
134             current_image = imgproc.getUndistortionImage(current_image); //
    Kinect のキャリブレーションを行い, キャリブレーション結果を適用する (c71)
135             imgproc.showImage(main_windowname, current_image);
136             //flip(current_image, flip_image, 1);
137             //imgproc.showImage("Original - Flip", flip_image); //Kinect から取得した画像を表示
138 
139             //タイヤも含めて前面の点群を取得する
140             namedWindow(param_windowname, CV_WINDOW_KEEPRATIO);
141             createTrackbar("Threshold(0-255)", param_windowname, &imgproc.th, 255);
142             createTrackbar("Neighborhood Level(0-10)", param_windowname, &imgproc.
    neighborhood, 10);
143             createTrackbar("Closing Times Level(0-10)", param_windowname, &imgproc.
    closing_times, 10);
144             bin_image = imgproc.getBackgroundSubstractionBinImage(current_image, background_gray_image/*,
     imgproc.th, imgproc.med, imgproc.cnt*/);
145             //ユニット部だけ切り取る (c77)
146             //bin_image = imgproc.getUnitMask(bin_image);
147             imgproc.showImage(maskbinimage_windowname, bin_image); //確認用に切り取った画像を表示する
148 
149             //ポイントクラウドの取得 (c57)
150             cloud = kinect.getPointCloud(bin_image); //ポイントクラウドの取得 (c57). 切り取った画像をもとにする
151             pointcloudlibrary.flagChecker(); //各点群処理のフラグをチェックするメソッド (c64)
152             cout << "
    ================================================================================" << endl;
153             cout << "Original PointCloud Size\t=>\t" << cloud->size() << endl;
154 
155             //PCL の処理
156             if (pointcloudlibrary.passthrough_flag == true){  //外れ値除去 (c59)
157                 cloud = pointcloudlibrary.passThroughFilter(cloud, "z", 400, 30000); //
    Kinect から取得した外れ値を除去 (c60). 与えた軸の中で自分が取得したい範囲の下限と上限を与えることでそれ以外を省く (c81)
158                 //cloud = pointcloudlibrary.radiusOutlierRemoval(cloud); //半径を指定して外れ値を除去 (c60)
159             }
160 
161             if (pointcloudlibrary.downsampling_flag == true){   //ダウンサンプリング処理 (c59)
162                 //cloud = pointcloudlibrary.downSamplingUsingVoxelGridFilter(pointcloudlibrary.cloud, 2.0f,
     2.0f, 2.0f); //Default=all 0.003
163                 cloud = pointcloudlibrary.downSamplingUsingVoxelGridFilter(cloud, 2.5f, 2.5f, 2.5f); //
    Default=all 0.003
164                 //pointcloudlibrary.cloud =
     pointcloudlibrary.downSamplingUsingVoxelGridFilter(pointcloudlibrary.cloud, 0.001, 0.001, 0.001); //Default=all 0.003
165             }
166 
167             if (pointcloudlibrary.statisticaloutlierremoval_flag == true){
168                 cloud = pointcloudlibrary.removeOutlier(cloud); //統計的な外れ値除去 (c60)
169             }
170 
171             if (pointcloudlibrary.downsampling_flag == true && pointcloudlibrary.mls_flag == true){  //
    スムージング処理 (c60)
172                 cloud = pointcloudlibrary.smoothingUsingMovingLeastSquare(cloud, true, true, 2.5); //0.002
     < radius < ○. 小さいほど除去される
173 
174             }
175             else if (pointcloudlibrary.downsampling_flag == false && pointcloudlibrary.mls_flag == true){
176                 cout << "MLS を有効にするためには, ダウンサンプリングを有効にしてください" << endl;
177             }
178 
179             if (pointcloudlibrary.extractplane_flag == true){   //平面検出とクラスタリング (c61)
180                 cloud = pointcloudlibrary.getExtractPlaneAndClustering(cloud, true, 100, 5, false, true, 0.
    02, /*350*/150, /*25000*//*20000*/200000); //Default=0.03(前処理なしの場合)
181             }
182 
183             pointcloudlibrary.centroid = pointcloudlibrary.getCentroidCoordinate3d(cloud); //重心座標の計算
184 
185 
186             coefficient_plane = lsm.getCoefficient(cloud); //最小二乗法を行い平面の係数 [a b c]' を取得する (c78)
187             attitude_angle = lsm.calcYawRollPitch(coefficient_plane); //姿勢角を取得 (c78)
```

```
188             //cout << "[Yaw, Roll, Pitch]" << attitude_angle.yaw << " , " << attitude_angle.roll << " , "
     << attitude_angle.pitch << endl;
189
190             ev3control.set6DoFEV3(cloud, pointcloudlibrary.centroid, attitude_angle); //6DoF をまとめる
191
192             //平均座標に球を描画する (c83)
193             pcl::PointXYZ sphere;
194             sphere.x = pointcloudlibrary.centroid.x; //平均座標の x 座標
195             sphere.y = pointcloudlibrary.centroid.y; //平均座標の y 座標
196             sphere.z = pointcloudlibrary.centroid.z; //平均座標の z 座標
197             pointcloudlibrary.visualizer->addSphere(sphere, 10, 0.5, 0.0, 0.0, "sphere"); //平均座標に球を描
画
198
199             //Kinect から取得した点群を描画
200             pointcloudlibrary.visualizer->addPointCloud(cloud, "show cloud"); //点群を描画
201             //visualizer->updatePointCloud<pcl::PointXYZRGB>(cloud, "show cloud");
202             cout << "
    ============================================================================================" << endl;
203
204             //pointcloudlibrary.viewer->showCloud(cloud); //処理後の点群を表示
205             pointcloudlibrary.visualizer->spinOnce(); //PCLVisualizer を描画
206             //終了のためのキー入力チェック兼表示のためのウェイトタイム
207             kinect.key = waitKey(1); //OpenCV のウインドウを表示し続ける
208
209             //PCL のフレームレートを計算する用 (c61)
210             sys.endTimer(); //タイマーを終了 (c65)
211             cout << sys.getProcessTimeinMiliseconds() << "[ms], " << sys.
    getFrameRate() << " fps" << "\n" << endl;
212
213             //キーが入力されていれば以下を実行する．GetAsyncKeyState を利用することで
214             if (GetAsyncKeyState('R')){
215                 system("cls");
216                 destroyAllWindows(); //PCL または，OpenCV 画面上で'q' キーが入力されたら OpenCV のウインドウを閉じて処
理を終了 (c66)
217                 //pointcloudlibrary.viewer->~CloudViewer(); //クラウドビューアーの削除
218                 sys.removeDirectory(); //再計測を行う場合は，現在のデータは必要ないため削除
219                 cout << "Data Removed." << endl;
220                 save_count = 0;
221                 system("cls"); //cmd をクリア
222                 cout << "RETRY" << endl;
223                 goto RETRY;
224             }
225             else if (GetAsyncKeyState('P')){ //その時点のデータを保存する．複数回データを計測する際はプログラムを起動し
なおす手間が省ける
226                 cout << "Save the Current Data." << endl;
227                 sys.saveDataEveryEnterKey(current_image,bin_image,ev3control.
    ev3_6dof, cloud);
228                 draw.gnuplotScriptEV3Unit(coefficient_plane); //gnuplot 用のスクリプト
229                 sys.save_flag = true; //6DoF 情報を出力するフラグをオンにする (c82)
230                 save_count++;
231             }
232             else if (GetAsyncKeyState('L')){ //'l' が入力されたら．EV3 軌道が欲しい時に入力する
233                 saveev3route_flag = true; //フラグを true にする
234             }
235
236             //'l' キーが入力されていれば，平均座標の軌道を追跡し続ける (c82)
237             if (saveev3route_flag == true){ //フラグが true であれば，平均座標の軌道を保存する (c82)
238                 sys.saveDataContinuously(ev3control.ev3_6dof);
239             }
240
241             //PCLVisualizer に描画した点群を削除する
242             pointcloudlibrary.visualizer->removePointCloud("show cloud");
243             pointcloudlibrary.visualizer->removeShape("sphere");
244
245             system("cls"); //コンソール内の表示をリセット (c64)
246         }
247
248         //計測が終了したところ (PCL 上，OpenCV ウインドウ上，コンソール上で'q' が押されてたとき)
249         destroyAllWindows(); //PCL または，OpenCV 画面上で'q' キーが入力されたら OpenCV のウインドウを閉じて処理を終了
(c66)
250         //pointcloudlibrary.viewer->~CloudViewer(); //クラウドビューアーの削除
251         pointcloudlibrary.visualizer->~PCLVisualizer(); //PCLVisualizer の削除
252         if (saveev3route_flag == true){ //一度でもデータを保存していれば，どちらかのフラグは true になる
253             draw.gnuplotScriptEV3Route(); //軌道をプロットするスクリプトを保存する
254         }
255
256         //データを保存するかの確認
257         if (saveev3route_flag == true || sys.save_flag == true){ //
    データを保存するフラグが true(=データが保存されている) なら保存するかどうか確認する
258             cout << "Save Data?" << endl;
259             int checkNum = sys.alternatives(); //'1' なら保存，'0' なら削除
260             if (checkNum == 1){
261                 sys.endMessage(checkNum);
262             }
263         }
264         else{
265             sys.removeDirectory(); //ディレクトリの削除
```

```
266             sys.endMessage();
267         }
268     }
269     catch (exception& ex){ //例外処理
270         cout << ex.what() << endl;
271         destroyAllWindows(); //OpenCV で作成したウインドウを全て削除する (c35)
272         //pointcloudlibrary.viewer->~CloudViewer(); //クラウドビューアーの削除
273         pointcloudlibrary.visualizer->~PCLVisualizer(); //PCLVisualizer の削除
274         //異常終了した時はデータを保存する必要がないので削除
275         sys.removeDirectory();
276         cout << "Data Removed." << endl;
277         return -1;
278     }
279     return 0;
280 }
```

4.11.1.2 void onMouse (int *event,* int *x,* int *y,* int *flags,* void * *param*)

マウス操作

Mouse.cpp の 12 行目に定義があります。

```
13 {
14     if (selectObject)
15     {
16         selection.x = MIN(x, origin.x);
17         selection.y = MIN(y, origin.y);
18         selection.width = abs(x - origin.x);
19         selection.height = abs(y - origin.y);
20         selection &= Rect(0, 0, image.cols, image.rows);
21     }
22
23     switch (event)
24     {
25     case CV_EVENT_LBUTTONDOWN:
26         origin = Point(x, y);
27         selection = Rect(x, y, 0, 0);
28         selectObject = true;
29         break;
30     case CV_EVENT_LBUTTONUP:
31         selectObject = false;
32         if (selection.width > 0 && selection.height > 0)
33         {
34             trackObject = -1;
35         }
36         break;
37     }
38
39     return;
40 }
```

4.11.2 変数詳解

4.11.2.1 char directoryName[NOC]

フォルダ名

main.cpp の 23 行目に定義があります。

参照元 System::endMessage(), Drawing::gnuplotScript(), Drawing::gnuplotScriptCoG(), Drawing::gnuplotScriptEV3Route(), Drawing::gnuplotScriptEV3Unit(), System::makeDirectory(), System::openDirectory(), System::outputAllData(), System::outputVideo(), Drawing::plot3D(), Drawing::plot3DRealTime(), System::removeDirectory(), System::saveDataContinuously(), System::saveDataEveryEnterKey().

4.11.2.2 Mat image

RGB 画像格納用の変数

main.cpp の 21 行目に定義があります。

参照元 Kinect::getPointCloud(), main(), onMouse().

4.11.2.3 Point origin

オリジナルの座標
main.cpp の 28 行目に定義があります。
参照元 onMouse().

4.11.2.4 int save_count = 0

main.cpp の 32 行目に定義があります。
参照元 Drawing::gnuplotScriptEV3Unit(), main(), System::saveDataEveryEnterKey().

4.11.2.5 Rect selection

選択
main.cpp の 29 行目に定義があります。
参照元 onMouse().

4.11.2.6 bool selectObject = false

オブジェクト選択
main.cpp の 26 行目に定義があります。
参照元 onMouse().

4.11.2.7 int trackObject = 0

追跡するオブジェクト
main.cpp の 27 行目に定義があります。
参照元 onMouse().

4.12 Mouse.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

関数

- void **onMouse** (int event, int x, int y, int flags, void *param)
  マウス操作

4.12.1 関数詳解

4.12.1.1 void onMouse (int *event,* int *x,* int *y,* int *flags,* void * *param*)

マウス操作
Mouse.cpp の 12 行目に定義があります。
参照先 image, origin, selection, selectObject, trackObject.

```
13 {
14     if (selectObject)
15     {
16         selection.x = MIN(x, origin.x);
17         selection.y = MIN(y, origin.y);
18         selection.width = abs(x - origin.x);
19         selection.height = abs(y - origin.y);
20         selection &= Rect(0, 0, image.cols, image.rows);
21     }
22
23     switch (event)
24     {
25     case CV_EVENT_LBUTTONDOWN:
26         origin = Point(x, y);
27         selection = Rect(x, y, 0, 0);
28         selectObject = true;
29         break;
30     case CV_EVENT_LBUTTONUP:
31         selectObject = false;
32         if (selection.width > 0 && selection.height > 0)
33         {
34             trackObject = -1;
35         }
36         break;
37     }
38
39     return;
40 }
```

4.13 PathTrackingAndInductionOfTheRobot.hpp ファイル

クラス

- struct **Point3**
- struct **outputData**
- struct **DoF**
- struct **AttitudeAngle**

型定義

- typedef struct **Point3 Point3ius**
- typedef struct **outputData outputData**
- typedef struct **DoF DoF6d**
- typedef struct **AttitudeAngle AttitudeAngle3d**

関数

- void **onMouse** (int event, int x, int y, int, void *)

  マウス操作

変数

- Mat **image**

  RGB 画像格納用の変数
- char **directoryName** [**NOC**]

  フォルダ名
- bool **selectObject**

  オブジェクト選択
- int **trackObject**

  追跡するオブジェクト
- Point **origin**

オリジナルの座標

- Rect **selection**

  選択

- int **save_count**

4.13.1 型定義詳解

4.13.1.1 typedef struct AttitudeAngle AttitudeAngle3d

4.13.1.2 typedef struct DoF DoF6d

4.13.1.3 typedef struct outputData outputData

4.13.1.4 typedef struct Point3 Point3ius

4.13.2 関数詳解

4.13.2.1 void onMouse (int *event,* int *x,* int *y,* int , void *)

マウス操作

Mouse.cpp の 12 行目に定義があります。

参照先 image, origin, selection, selectObject, trackObject.

```
13 {
14     if (selectObject)
15     {
16         selection.x = MIN(x, origin.x);
17         selection.y = MIN(y, origin.y);
18         selection.width = abs(x - origin.x);
19         selection.height = abs(y - origin.y);
20         selection &= Rect(0, 0, image.cols, image.rows);
21     }
22
23     switch (event)
24     {
25     case CV_EVENT_LBUTTONDOWN:
26         origin = Point(x, y);
27         selection = Rect(x, y, 0, 0);
28         selectObject = true;
29         break;
30     case CV_EVENT_LBUTTONUP:
31         selectObject = false;
32         if (selection.width > 0 && selection.height > 0)
33         {
34             trackObject = -1;
35         }
36         break;
37     }
38
39     return;
40 }
```

4.13.3 変数詳解

4.13.3.1 char directoryName[NOC]

フォルダ名

main.cpp の 23 行目に定義があります。

参照元 System::endMessage(), Drawing::gnuplotScript(), Drawing::gnuplotScriptCoG(), Drawing::gnuplotScriptEV3Route(), Drawing::gnuplotScriptEV3Unit(), System::makeDirectory(), System::openDirectory(), System::outputAllData(), System::outputVideo(), Drawing::plot3D(), Drawing::plot3DRealTime(), System::removeDirectory(), System::saveDataContinuously(), System::saveDataEveryEnterKey().

4.13.3.2 Mat image

RGB 画像格納用の変数

main.cpp の 21 行目に定義があります。

参照元 Kinect::getPointCloud(), main(), onMouse().

4.13.3.3 Point origin

オリジナルの座標

main.cpp の 28 行目に定義があります。

参照元 onMouse().

4.13.3.4 int save_count

main.cpp の 32 行目に定義があります。

参照元 Drawing::gnuplotScriptEV3Unit(), main(), System::saveDataEveryEnterKey().

4.13.3.5 Rect selection

選択

main.cpp の 29 行目に定義があります。

参照元 onMouse().

4.13.3.6 bool selectObject

オブジェクト選択

main.cpp の 26 行目に定義があります。

参照元 onMouse().

4.13.3.7 int trackObject

追跡するオブジェクト

main.cpp の 27 行目に定義があります。

参照元 onMouse().

4.14 PointCloudLibrary.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "PointCloudLibrary.hpp"
```

4.15 PointCloudLibrary.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **PointCloudLibrary**

  点群処理を行うクラス

4.16 stdafx.cpp ファイル

4.17 stdafx.h ファイル

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>
#include <math.h>
#include <ctype.h>
#include <sys/stat.h>
#include <Windows.h>
#include <mmsystem.h>
#include <NuiApi.h>
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\video\tracking.hpp>
#include <opencv2\flann\flann.hpp>
#include <pcl\point_types.h>
#include <pcl\point_cloud.h>
#include <pcl\io\io.h>
#include <pcl\io\pcd_io.h>
#include <pcl\io\ply_io.h>
#include <pcl\common\common_headers.h>
#include <pcl\visualization\cloud_viewer.h>
#include <pcl\visualization\pcl_visualizer.h>
#include <pcl\filters\passthrough.h>
#include <pcl\filters\statistical_outlier_removal.h>
#include <pcl\filters\radius_outlier_removal.h>
#include <pcl\kdtree\kdtree_flann.h>
#include <pcl\surface\mls.h>
#include <pcl\filters\voxel_grid.h>
#include <pcl\ModelCoefficients.h>
#include <pcl\sample_consensus\method_types.h>
#include <pcl\sample_consensus\model_types.h>
#include <pcl\segmentation\sac_segmentation.h>
#include <pcl\filters\extract_indices.h>
#include <pcl\features\normal_3d.h>
#include <pcl\segmentation\extract_clusters.h>
#include <boost\thread\thread.hpp>
#include <pcl\registration\icp.h>
#include <pcl\PCLPointField.h>
```

マクロ定義

- #define **_CRT_SECURE_NO_WARNINGS**
- #define **WIDTH** 640

  ＜名前空間
- #define **HEIGHT** 480

  画像の高さ
- #define **ALLPIXEL WIDTH∗HEIGHT**

  1 フレームの全ピクセル数
- #define **NOC** 64

  Number of Characters. (ファイルの名前を付けるときの文字数制限)
- #define **OUTPUTDATA_MAX** 10000

  出力するデータの上限

4.17.1 マクロ定義詳解

4.17.1.1 #define _CRT_SECURE_NO_WARNINGS

stdafx.h の 8 行目に定義があります。

4.17.1.2 #define ALLPIXEL WIDTH∗HEIGHT

1 フレームの全ピクセル数

stdafx.h の 78 行目に定義があります。

4.17.1.3 #define HEIGHT 480

画像の高さ

stdafx.h の 77 行目に定義があります。

4.17.1.4 #define NOC 64

Number of Characters. (ファイルの名前を付けるときの文字数制限)

stdafx.h の 79 行目に定義があります。

参照元 Drawing::gnuplotScript(), Drawing::gnuplotScriptCoG(), Drawing::gnuplotScriptEV3Route(), Drawing::gnuplotScriptEV3Unit(), System::makeDirectory(), System::openDirectory(), System::outputAllData(), System::outputVideo(), Drawing::plot3DRealTime(), System::removeDirectory(), System::saveDataContinuously(), System::saveDataEveryEnterKey().

4.17.1.5 #define OUTPUTDATA_MAX 10000

出力するデータの上限

stdafx.h の 80 行目に定義があります。

4.17.1.6 #define WIDTH 640

＜名前空間

画像の幅

stdafx.h の 76 行目に定義があります。

4.18 System.cpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
#include "System.hpp"
```

4.19 System.hpp ファイル

```
#include "PathTrackingAndInductionOfTheRobot.hpp"
```

クラス

- class **System**

Index