```
/*
	ICPスキャンマッチングによるSLAMの実行









*/
//*****************SLAM関連の定数マクロ*************************************************************************
#define Grid_Size	100					//格子のサイズ(縦横同じ)
#define MAP_Size_X	30000				//地図全体のサイズ[mm]
#define MAP_Size_Y	30000				//地図全体のサイズ[mm]
#define Grid_NUM_X MAP_Size_X/Grid_Size //地図全体の格子の数 X
#define Grid_NUM_Y MAP_Size_Y/Grid_Size //地図全体の格子の数 Y
#define Kaisu_MD 20					//ICP価値関数最小化の回数
#define Thred_N  10					//占有と判定する点の数(N個以上あれば占有とみなす)
#define Thred_D  800				//これ以上離れていたら対応点を結ばない
#define dthreS    50				//等間隔リサンプリングの点間隔
#define dthreR   500				//等間隔サンプリングを実施する，台車からの最低距離
#define dthreT   500				//等間隔リサンプリングのMAX間隔(これ以上離れたら補完しない)
#define dLimit   700				//Lidar距離のリミット(自分自身のスキャンの排除)
#define Kaiseki_Mode 1				//0***直線距離    1***垂直距離
#define MAX_SCAN_NUM 1000			//スキャンの最大数
#define MAX_ONE_SCAN 600			//1スキャンの最大データ数
#define PLUS_MAX_ONE_SCAN 1000		//補完後の全スキャンの数(注意！リサンプリング後のスキャンサイズ)
#define ITMAX   100					/* 反復回数の上限 */
#define CGOLD   0.381966			/* 黄金分割比 */
#define ZEPS    1.0E-10				/* 極小がちょうど x=0 にあるときは相対精度 tol */
                                    /* の代わりにこれを絶対精度とする */
#define SHFT(a,b,c,d)    (a)=(b); (b)=(c); (c)=(d);
#define SIGN(a, b)              ((b) >= 0.0? fabs(a): -fabs(a))
#define PI 3.141592
typedef struct{
                float x;
                float y;
} Zahyo;
Zahyo     Measure_scan[PLUS_MAX_ONE_SCAN];
Zahyo     Measure_scan2[PLUS_MAX_ONE_SCAN];
typedef struct{
                double x;
                double y;
                double c;
} OPT_TRANS;
OPT_TRANS       ROBO_POS_G;                                     //現在の台車の位置姿勢
OPT_TRANS   ROBO_POS[MAX_SCAN_NUM];                         //各SCAN毎の台車の位置姿勢
OPT_TRANS       ROBO_POS_2[MAX_SCAN_NUM];                   //各SCAN毎の台車の位置姿勢(ログ取り用)
OPT_TRANS       OPT_XYC;                                        //点群の最適移動量
OPT_TRANS       OPT_XYC_SUM;                                //点群の最適移動量(総和)
OPT_TRANS       OPT_XYC_M[MAX_SCAN_NUM];
int             SCAN_DATA_NUM[MAX_SCAN_NUM];                //1スキャンのデータ数
int             SCAN_DATA_NUM2[MAX_SCAN_NUM];               //1スキャンのデータ数(補完点列)
int             SCAN_NUM_ALL=0;                                 //全スキャンの数
int             Measure_Num=0;                                  //1スキャンのデータ数
Zahyo   Scan_Data[MAX_SCAN_NUM][MAX_ONE_SCAN];          //スキャンごとの点列座標=2x4x1000x360=2.88MByte

typedef struct{
                int num;                                    //グリッド内に存在する点の数
                int flag;                                   //障害物があるか無いかの判別フラグ
                float nx;                                   //グリッドを代表する法線ベクトル
                float ny;                                   //グリッドを代表する法線ベクトル
                float gx;                                   //グリッド内の点の重心座標
                float gy;                                   //グリッド内の点の重心座標
} GridData;
GridData G_MAP_1[Grid_NUM_X][Grid_NUM_Y];                   //グリッドマップデータ   6x4x300x300=2.16MByte

typedef struct{
        float                   x;                          //現在位置[mm]
        float                   y;                          //現在位置[mm]
        float                   c;                          //現在角度[deg]
        float                   gx;                         //目標位置[mm]
        float                   gy;                         //目標位置[mm]
} car_pos;
car_pos CAR_POS;
car_pos CAR_POS_M[100000];      //100sec間のデータを保持できる 7x4x100000=2.8MByte
int             OPT_XYC_NUM=0;                                  //移動回数
Zahyo   Gnp;                                                    //中間変数
float   Gdis;                                                   //中間変数
Zahyo   Scan_Normal[PLUS_MAX_ONE_SCAN];                 //各点における法線ベクトル

OPT_TRANS Gpose;                                                //中間変数
```

```
84      OPT_TRANS Gdp;                                                          //中間変数
85
86      Zahyo   Pair_Grid_1[PLUS_MAX_ONE_SCAN];                         //ペアグリッドの対応点

87      int Iteration_Count=0;                                                  //繰り返し回数
88      float SENSOR_GENTEN_X2=   0.00;                                 //センサーと車軸中心のオフセット

89      float SENSOR_GENTEN_Y2= 495.00;                                 //センサーと主軸中心のオフセット

90
91      Zahyo START_POS;
92      Zahyo GOAL_POS;
93      Zahyo Trans_POS;
94      int     Pm_CNT=0;
95      int     Pm_CNT_Total=0;
96      Zahyo SP_T_POS[100];                                                    //スプラインの通過座標

97      int   SP_T_NUM=0;
98
99      char FileName_GridMap[255]="TMap_1.txt";                //ロングファイル名禁止

100     char FileName_RoboPos[255]="TPos_1.txt";                //ロングファイル名禁止

101
102     char FileName_PassMap[255]="TMap_2.txt";                //ロングファイル名禁止

103     char FileName_PassPos[255]="TPos_2.txt";                //ロングファイル名禁止

104
105     char FileName_GoalPos[255]="Goal.txt";                  //ロングファイル名禁止

106
107     int Get_Dis(Zahyo a,Zahyo b);
108     float Get_Dis_F(Zahyo a,Zahyo b);

109     int Find_InterpolatePoint(Zahyo cp,Zahyo pp,float re_sample_width,int *inserted);

110     int Resample_Point(int ref_data,float re_sample_width);

111     Zahyo calNormal(int dir,int idx,int ref_data,Zahyo scan[]);

112     void Get_Normal(int ref_data,Zahyo scan[]);

113     void Transform_OR(float xt,float yt,float c);

114     void Init_Grid_Map(void);
115     void Get_Grid_Index(Zahyo p1,int *x,int *y);

116     Zahyo Get_Grid_Zahyo(int i,int j);

117     void Add_Points_to_GridMap_2(void);

118     float Get_Dis_Grid(Zahyo a,int *i,int *j);

119     void Get_TaiouTen_Grid(int L_thre);

120     double Get_Ev_Grid_ED(double xt,double yt,double ct);

121     double Get_Ev_Grid_PD(double xt,double yt,double ct);

122     double brent(double ax, double bx, double cx, double (*f)(double),double tol, double *xmin);

123     double f(double tt);
124     double serch2(void);
125     int Get_Opt_Trans_4(void);
126     float add_angle(float a1, float a2);

127     int ICP_Grid(int idx);
128     void Test_Auto_Drive_1(void);
129     void Test_Auto_Drive_2(void);
130
131
132
133     int Get_Dis(Zahyo a,Zahyo b)
134     {
135             int L;
136             L=sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));

137             return L;
138     }
139     float Get_Dis_F(Zahyo a,Zahyo b)

140     {
141             float L;
142             L=sqrtf((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
143             return L;
144     }
145
146     int Find_InterpolatePoint(Zahyo cp,Zahyo pp,float re_sample_width,int *inserted)

147     {
148             int flag;
149             float L;
150             L=Get_Dis_F(cp,pp);                     //2座標間の距離を計算

151             *inserted=0;
152     //**********点を削除する場合**************************

153             if(L+Gdis<re_sample_width){
154             flag=0;                                                 //flag=0で点を削除
155             Gdis+=L;
156             }
157     //**********距離が離れすぎている場合→補完点を追加しない***************

158             else if(L+Gdis>=dthreT){
159             flag=1;                                                 //flag=1で点を挿入
160             Gnp.x=cp.x;                                             //現在の座標を登録
161             Gnp.y=cp.y;                                             //現在の座標を登録
162             }
163     //***********点を追加する場合*******************************************

164             else{
165             flag=1;
166             *inserted=1;
167             Gnp.x=pp.x+(cp.x-pp.x)*(re_sample_width-Gdis)/L;                 //点を追加

168             Gnp.y=pp.y+(cp.y-pp.y)*(re_sample_width-Gdis)/L;                 //点を追加

169             }
170             return flag;
171     }
172     /*
173             スキャン点均一化
174
```

```
175     */
176     int Resample_Point(int ref_data,float re_sample_width)
177     {
178             int i,k;
179             float L,r;
180             int inserted,flag;
181             Zahyo lp,prevLP;
182
183             lp.x    =Measure_scan[0].x;
184             lp.y    =Measure_scan[0].y;
185             prevLP.x=lp.x;                                          //最初の点をコピー代入しておく
186             prevLP.y=lp.y;                                          //最初の点をコピー代入しておく
187             Gnp.x   =lp.x;
188             Gnp.y   =lp.y;
189             Gdis=0.0;
190             k=0;
191             Measure_scan2[0].x=Measure_scan[0].x;   //最初の点をコピー代入しておく
192             Measure_scan2[0].y=Measure_scan[0].y;   //最初の点をコピー代入しておく
193
194             for(i=1;i<ref_data;i++){
195             inserted=0;
196             lp.x=Measure_scan[i].x;
197             lp.y=Measure_scan[i].y;
198             r=sqrtf(lp.x*lp.x+lp.y*lp.y);                   //測定点までの距離
199             Measure_scan2[i].x=lp.x;
200             Measure_scan2[i].y=lp.y;
201             flag=Find_InterpolatePoint(lp,prevLP,re_sample_width,inserted);
202     //***************追加点ありの場合*******************************************
203             if(flag==1){
204             if(r>dthreR){
205             Measure_scan2[ref_data+k].x=Gnp.x;                      //新しい点を追加
206             Measure_scan2[ref_data+k].y=Gnp.y;                      //新しい点を追加
207             k++;
208             }
209             prevLP.x=Gnp.x;                                         //ひとつ前の点を更新(挿入)
210             prevLP.y=Gnp.y;                                         //ひとつ前の点を更新(挿入)
211             Gdis=0;
212             if(inserted==1) i--;
213             }
214     //*****************追加点なし(点を削除)の場合*****************************
215             else{
216             prevLP.x=lp.x;                                          //
217             prevLP.y=lp.y;                                          //
218             }
219             }
220             return ref_data+k-1;
221     }
222
223     /*
224             法線ベクトルの計算
225
226     */
227     Zahyo calNormal(int dir,int idx,int ref_data,Zahyo scan[])
228     {
229             int i;
230             float dx,dy,d;
231             Zahyo cp,lp,normal;
232             float d_max,d_min;
233             d_max=600;                                      //法線ベクトルを取得できる点間距離の最大値
234             d_min=60;                                       //法線ベクトルを取得できる点間距離の最小値
235             cp.x=scan[idx].x;
236             cp.y=scan[idx].y;
237             normal.x=0;                                     //法線ベクトルが求まらなかった場合の初期値
238             normal.y=0;                                     //法線ベクトルが求まらなかった場合の初期値
239             for(i=idx+dir;i>=0&&i<ref_data;i+=dir){
240             lp.x=scan[i].x;
241             lp.y=scan[i].y;
242             dx=lp.x-cp.x;
243             dy=lp.y-cp.y;
244             d=sqrtf(dx*dx + dy*dy);
245             if(d>d_min&&d<d_max){
246             normal.x=dy/d;                                  //法線ベクトル
247             normal.y=-dx/d;                                 //法線ベクトル
248             break;
249             }
250             if(d>=d_max) break;
251             }
252             return normal;
253     }
254     /*
255             現状のスキャンに対して法線ベクトルを求める
256
257     */
258     void Get_Normal(int ref_data,Zahyo scan[])
259     {
260             int i,flag;
261             Zahyo normal;
262             Zahyo nL,nR;
263             float dx,dy,L;
264             for(i=0;i<ref_data;i++){
265             nL=calNormal(-1,i,ref_data,scan);
266             nR=calNormal( 1,i,ref_data,scan);
267     //**********ベクトルの向きをそろえる(正の側にそろえる)***************
268             nR.x=-nR.x;
269             nR.y=-nR.y;
270     //***********両側ベクトルの平均を取る*********************************
271             if(nL.x==0&&nL.y==0){
272             normal.x=nR.x;
273             normal.y=nR.y;
274             }
```

```
275             else if(nR.x==0&&nR.y==0){
276             normal.x=nL.x;
277             normal.y=nL.y;
278             }
279             else if((nL.x==0&&nL.y==0)&&(nR.x==0&&nR.y==0)){

280             normal.x=0;
281             normal.y=0;
282             }
283             else{
284             dx=nL.x+nR.x;
285             dy=nL.y+nR.y;
286             L=sqrtf(dx*dx + dy*dy);
287             if(L==0){
288             normal.x=0;
289             normal.y=0;
290             }
291             else{
292             normal.x=dx/L;
293             normal.y=dy/L;
294             }
295             }
296             Scan_Normal[i].x=normal.x;
297             Scan_Normal[i].y=normal.y;
298             }
299     }
300
301
302     //*********参照座標をx,y平行移動しc回転させる*******************

303     void Transform_OR(float xt,float yt,float c)

304     {
305             int i;
306             float x,y,c_rad;
307             float xg,yg;
308             for(i=0;i<Measure_Num;i++){
309             if((Measure_scan[i].x==0&&Measure_scan[i].y==0)){               //(0,0)のエラー点, 対応不可点は移動させない

310             Measure_scan[i].x=0;                                                    //0のまま
311             Measure_scan[i].y=0;                                                    //0のまま
312             }
313             else{
314             x=(float)Measure_scan[i].x;                                     //移動元のX座標

315             y=(float)Measure_scan[i].y;                                     //移動元のY座標

316             c_rad=c/180.0*PI;                                                       //角度をradに変換

317             xg=x*cosf(c_rad)-y*sinf(c_rad)+xt;                              //回転行列・並進行列をかける

318             yg=x*sinf(c_rad)+y*cosf(c_rad)+yt;                              //回転行列・並進行列をかける

319             Measure_scan[i].x=xg;                                           //新しい参照点を代入

320             Measure_scan[i].y=yg;                                           //新しい参照点を代入

321             }
322             }
323     //**************[0,0]点の移動と回転***************************************

324             x=Trans_POS.x;                                                  //移動元のX座標
325             y=Trans_POS.y;                                                  //移動元のY座標
326             c_rad=c/180.0*PI;                                               //角度をradに変換

327             xg=x*cosf(c_rad)-y*sinf(c_rad)+xt;                              //回転行列・並進行列をかける

328             yg=x*sinf(c_rad)+y*cosf(c_rad)+yt;                              //回転行列・並進行列をかける

329             Trans_POS.x=xg;                                                 //移動後のX座標
330             Trans_POS.y=yg;                                                 //移動後のY座標
331     }
332
333     //**************グリッドマップを初期化する*********************************

334     void Init_Grid_Map(void)
335     {
336             int i,j;
337             for(i=0;i<Grid_NUM_X;i++){
338                     for(j=0;j<Grid_NUM_Y;j++){
339                     G_MAP_1[i][j].num=0;                    //点の数をゼロリセット

340                     G_MAP_1[i][j].flag=0;                   //占有フラグをクリア

341                     G_MAP_1[i][j].nx=0;                             //法線ベクトルを０にセット

342                     G_MAP_1[i][j].ny=0;                             //法線ベクトルを０にセット

343                     G_MAP_1[i][j].gx=0;                             //重心座標を０にセット

344                     G_MAP_1[i][j].gy=0;                             //重心座標を０にセット

345                     }
346             }
347     }
348
349     /*
350             現在座標に対するグリッドのインデックス番号を返す

351
352
353     */
354     void Get_Grid_Index(Zahyo p1,int *x,int *y)

355     {
356             *x=floor(p1.x/Grid_Size+0.5)+0.5*Grid_NUM_X;

357             *y=floor(p1.y/Grid_Size+0.5)+0.5*Grid_NUM_Y;

358             if(*x>=Grid_NUM_X) *x=Grid_NUM_X-1;

359             if(*y>=Grid_NUM_Y) *y=Grid_NUM_Y-1;

360     }
361     /*
362             インデックス指定したグリッドの中心座標を返す

363
364     */
365     Zahyo Get_Grid_Zahyo(int i,int j)

366     {
367             Zahyo p1;
368             p1.x=Grid_Size*i-Grid_Size*Grid_NUM_X/2;
```

```
369             p1.y=Grid_Size*j-Grid_Size*Grid_NUM_Y/2;

370             return p1;
371     }
372
373     /*
374             点列データをグリッドマップに登録

375
376     */
377
378     //**************位置合わせが終わった観測データをマップに追加する********************

379     void Add_Points_to_GridMap_2(void)

380     {
381             int i;
382             int x,y;
383             float u,n;
384             Zahyo p1;
385
386             for(i=0;i<Measure_Num;i++){
387             p1=Measure_scan[i];                                             //スキャンデータの注目点の座標

388             if(p1.x==0&&p1.y==0){}
389             else{
390             Get_Grid_Index(p1,x,y);                                 //p1に対応するグリッドのインデックス番号を呼び出す

391             G_MAP_1[x][y].num+=1;                                   //点の個数をカウントアップ

392
393             n=(float)G_MAP_1[x][y].num;                             //データの個数

394     //      if(n>MAX_NUM) MAX_NUM=n;
395
396             if(n>=Thred_N){
397             G_MAP_1[x][y].flag=1;                                   //点の数が多い
398             }
399             else
400             G_MAP_1[x][y].flag=0;                                   //点の数が少ない→占有してない

401
402             u=G_MAP_1[x][y].nx;                                             //前回までの平均値

403             G_MAP_1[x][y].nx=u-u/n+Scan_Normal[i].x/n;              //法線ベクトルの平均値(逐次平均)

404             u=G_MAP_1[x][y].ny;                                             //前回までの平均値

405             G_MAP_1[x][y].ny=u-u/n+Scan_Normal[i].y/n;              //法線ベクトルの平均値(逐次平均)

406
407             u=G_MAP_1[x][y].gx;                                             //前回までの平均値

408             G_MAP_1[x][y].gx=u-u/n+p1.x/n;                                  //重心座標の平均値(逐次平均)

409             u=G_MAP_1[x][y].gy;                                             //前回までの平均値

410             G_MAP_1[x][y].gy=u-u/n+p1.y/n;                                  //重心座標の平均値(逐次平均)

411             }
412             }
413     }
414     //**********ある座標と対応する近傍グリッドの近傍点との距離*****************************

415     float Get_Dis_Grid(Zahyo a,int *i,int *j)

416     {
417             int t,u,t_min,u_min;
418             int k,flag;
419             int xx,yy,R;
420             float L,L_min;
421             float x,y;
422
423             L_min=999999999;
424             t_min=-1;
425             u_min=-1;
426             k=0;
427             R=Thred_D/Grid_Size;                                    //対応点から半径Rの範囲(これより遠い点は除外する)

428
429     //      for(t=0;t<Grid_NUM_X;t++){
430     //              for(u=0;u<Grid_NUM_Y;u++){
431             for(xx=-R;xx<=R;xx++){
432                     for(yy=-R;yy<=R;yy++){
433
434                     t=*i+xx;
435                     u=*j+yy;
436
437                     if((t>=0&&t<Grid_NUM_X)&&(u>=0&&u<Grid_NUM_Y)){

438                     x=G_MAP_1[t][u].gx;                     //グリッド内の点の重心座標

439                     y=G_MAP_1[t][u].gy;                     //グリッド内の点の重心座標

440                     flag=G_MAP_1[t][u].flag;        //占有フラグ

441                     }
442                     else{
443                     x=0;                                            //グリッドの外なので点が存在しないものとする

444                     y=0;                                            //グリッドの外なので点が存在しないものとする

445                     }
446
447                     if((x==0&&y==0)||flag==0)
448                     L=9999999;                                                      //点が存在しない場合距離は大きな値

449                     else
450                     L=sqrtf((a.x-x)*(a.x-x)+(a.y-y)*(a.y-y));       //点が存在する場合

451
452                     if(L<L_min){
453                     L_min=L;                                                        //最短距離
454                     t_min=t;                                                        //最短距離の時のグリッドインデックス

455                     u_min=u;                                                        //最短距離の時のグリッドインデックス

456                     }
457                     k++;
458             }
459
460     }
461
462     *i=t_min;                                               //最短距離の時のグリッドインデックス
```

```
463             *j=u_min;                                                       //最短距離の時のグリッドインデックス
464             return L_min;                                           //最短距離
465     }
466
467     //***************Measure_scan[]に対するグリッドマップの対応点を求める**********************
468     void Get_TaiouTen_Grid(int L_thre)
469     {
470             int i;
471             int x,y;
472             int L;
473     //************対応点(最小距離にある点)を見つける*****************************
474             for(i=0;i<Measure_Num;i++){
475             if(Measure_scan[i].x==0&&Measure_scan[i].y==0){
476             Pair_Grid_1[i].x=-1;                                            //現在スキャン点がエラー点の場合
477             Pair_Grid_1[i].y=-1;                                            //対応点もエラーに落とす
478             }
479             else{
480             Get_Grid_Index(Measure_scan[i],x,y);            //現在座標に対するグリッドのインデックス番号を得る
481             L=Get_Dis_Grid(Measure_scan[i],x,y);            //現在座標の周辺半径Rマスの最小距離点を調べる
482
483
484             if(L>L_thre){
485             x=-1;                                                                   //点が離れすぎている場合，対応点無しの印
486             y=-1;                                                                   //点が離れすぎている場合，対応点無しの印
487             }
488
489             Pair_Grid_1[i].x=x;
490             Pair_Grid_1[i].y=y;
491             }
492
493             }
494     }
495
496     /*
497             Measure_scan[]を移動させてMAP_Grid[]との直線距離の2乗平均を返す
498
499     */
500     double Get_Ev_Grid_ED(double xt,double yt,double ct)
501     {
502
503             int i,j,z_cnt;
504             int xx,yy;
505             double x,y,c_rad;
506             double xg,yg;
507             Zahyo REF[PLUS_MAX_ONE_SCAN];
508             double dx,dy,r;
509
510
511             for(i=0;i<Measure_Num;i++){
512             x=(double)Measure_scan[i].x;                                    //移動元のX座標
513             y=(double)Measure_scan[i].y;                                    //移動元のY座標
514             if(x==0&&y==0){
515             REF[i].x=0;
516             REF[i].y=0;
517             }
518             else{
519             c_rad=PI*ct/180.0;                                                      //角度をradに変換
520             xg=x*cos(c_rad)-y*sin(c_rad)+xt;                        //回転行列・並進行列をかける
521             yg=x*sin(c_rad)+y*cos(c_rad)+yt;                        //回転行列・並進行列をかける
522             REF[i].x=xg;                                                            //新しい参照点を代入
523             REF[i].y=yg;                                                            //新しい参照点を代入
524             }
525             }
526
527             r=0;
528             z_cnt=0;
529             for(i=0;i<Measure_Num;i++){
530             xx=Pair_Grid_1[i].x;                                            //ペアのグリッドインデックス
531             yy=Pair_Grid_1[i].y;                                            //ペアのグリッドインデックス
532
533             if((REF[i].x==0&&REF[i].y==0)||(xx==-1&&yy==-1)){        //(0,0)のエラー点，対応不可点は無視する
534             z_cnt++;
535             }
536             else{
537             xg=G_MAP_1[xx][yy].gx;
538             yg=G_MAP_1[xx][yy].gy;
539
540             dx=REF[i].x-xg;
541             dy=REF[i].y-yg;
542
543             r+=dx*dx+dy*dy;                                                 //対応点の距離の総和
544             }
545
546             }
547             if((Measure_Num-z_cnt)==0) r=99999999;
548             else
549             r/=(double)(Measure_Num-z_cnt);                         //点の数で割って平均化する
550             return r;
551     }
552     /*
553             Measure_scan[]を移動させてMAP_Grid[]との垂直距離の2乗平均を返す
554
555     */
556     double Get_Ev_Grid_PD(double xt,double yt,double ct)
557     {
558             int xx,yy;
559             int i,j,z_cnt;
560             double x,y,c_rad;
561             double xg,yg,er;
562             Zahyo REF[PLUS_MAX_ONE_SCAN];
```

```
563             float nx,ny;
564             double dx,dy,r;
565
566
567             for(i=0;i<Measure_Num;i++){
568             x=(double)Measure_scan[i].x;                                    //移動元のX座標
569             y=(double)Measure_scan[i].y;                                    //移動元のY座標
570             if(x==0&&y==0){
571             REF[i].x=0;
572             REF[i].y=0;
573             }
574             else{
575             c_rad=PI*ct/180.0;                                                      //角度をradに変換
576             xg=x*cos(c_rad)-y*sin(c_rad)+xt;                        //回転行列・並進行列をかける
577             yg=x*sin(c_rad)+y*cos(c_rad)+yt;                        //回転行列・並進行列をかける
578             REF[i].x=xg;                                                            //新しい参照点を代入
579             REF[i].y=yg;                                                            //新しい参照点を代入
580             }
581
582             }
583
584             r=0;
585             z_cnt=0;
586             for(i=0;i<Measure_Num;i++){
587             xx=Pair_Grid_1[i].x;                                                    //ペアのグリッドインデックス
588             yy=Pair_Grid_1[i].y;                                                    //ペアのグリッドインデックス
589
590             if((REF[i].x==0&&REF[i].y==0)||(xx==-1&&yy==-1)){        //(0,0)のエラー点，対応不可点は無視する
591             z_cnt++;
592             }
593             else{
594             xg=G_MAP_1[xx][yy].gx;
595             yg=G_MAP_1[xx][yy].gy;
596             nx=G_MAP_1[xx][yy].nx;
597             ny=G_MAP_1[xx][yy].ny;
598
599             dx=REF[i].x-xg;
600             dy=REF[i].y-yg;
601
602             er=dx*nx+dy*ny;                                                                 //対応点の垂直距離
603             er*=er;                                                                                 //2乗する
604             r+=er;                                                                                  //対応点の距離の総和
605             }
606
607             }
608             if((Measure_Num-z_cnt)==0) r=99999999;
609             else
610             r/=(double)(Measure_Num-z_cnt);                                 //点の数で割って平均化する
611             return r;
612     }
613
614     /*
615     関数 f と、極小を囲い込む３点の座標 ax,bx,cx とを与えると、brent の方法で極小
616     を求める。与える３点の条件は、bx が ax,cx の間にあり、f(bx) が f(ax),f(bx) の
617     どちらよりも小さいことである。戻り値は極小値で、その x 座標は xmin に入る。
618     xmin の相対精度はほぼ tol になる。
619     */
620     double brent(double ax, double bx, double cx, double (*f)(double),double tol, double *xmin)
621     {
622             int iter;
623             double a, b, d, etemp, fu, fv, fw, fx, p, q, r, tol1, tol2, u, v, w, x, xm;
624             double e = 0.0;                                         /* 前々回の更新量 */
625
626             if(ax < cx)                                                     /* a < b にする */
627             {
628                     a = ax;
629                     b = cx;
630             }
631             else
632             {
633                     a = cx;
634                     b = ax;
635             }
636             x = w = v = bx;                                         /* 初期化 */
637             fw = fv = fx = (*f)(x);
638             for(iter = 1; iter <= ITMAX; iter++)    /* 主ループ */
639             {
640                     xm = 0.5 * (a + b);
641                     tol1 = tol * fabs(x) + ZEPS;
642                     tol2 = 2.0 * tol1;
643                     if(fabs(x - xm) <= (tol2 - 0.5 * (b - a)))      /* 収束判定 */
644                     {
645                             *xmin = x;                                      /* 最良の値を返す */
646                             return fx;
647                     }
648                     if(fabs(e) > tol1)                              /* 放物線補間してみる */
649                     {
650                             r = (x - w) * (fx - fv);
651                             q = (x - v) * (fx - fw);
652                             p = (x - v) * q - (x - w) * r;
653                             q = 2.0 * (q - r);
654                             if(q > 0.0)     p = -p;
655                             q = fabs(q);
656                             etemp = e;
657                             e = d;
658                             if(fabs(p) >= fabs(0.5 * q * etemp)
659                                     || p <= q * (a - x)
660                                     || p >= q * (b - x))    /* 放物線補間の適否の検査 */
661                             {
662                                     e = (x >= xm)? a - x: b - x;
663                                     d = CGOLD * e;                  /* 放物線補間は不適。大きい方の区間を */
664                                                                                     /* 黄金分割 */
665                             }
```

```
                else
                {
                        d = p / q;                                      /* 放物線補間を採択する */

                        u = x + d;
                        if(u - a < tol2 || b - u < tol2)        d = SIGN(tol1, xm - x);

                }
        }
        else
        {
                e = (x >= xm)? a - x: b - x;
                d = CGOLD * e;
        }
        u = x + ((fabs(d) >= tol1)? d: SIGN(tol1, d));

        fu = (*f)(u);                                           /* 主ループでの関数値評価はここだけ */

        if(fu <= fx)
        {
                if(u >= x)      a = x;
                else            b = x;
                SHFT(v, w, x, u);
                SHFT(fv, fw, fx, fu);
        }
        else
        {
                if(u < x)       a = u;
                else            b = u;
                if(fu <= fw || w == x)
                {
                        v = w;
                        w = u;
                        fv = fw;
                        fw = fu;
                }
                else if(fu <= fv || v == x || v == w)

                {
                        v = u;
                        fv = fu;
                }
        }
//      fprintf(stderr, "Error : Too many iterations in brent.¥n");

        *xmin = x;
        return fx;
}

double f(double tt)
{
        double tx,ty,tc;
        double v;
        tx=Gpose.x+tt*Gdp.x;
        ty=Gpose.y+tt*Gdp.y;
        tc=Gpose.c+tt*Gdp.c;
        if(Kaiseki_Mode==0)
        v=Get_Ev_Grid_ED(tx,ty,tc);
        else
        v=Get_Ev_Grid_PD(tx,ty,tc);

        return v;
}
double serch2(void)
{
        double t,v;
        double ax,bx,cx;
        double tol;

        ax=-2.0;
        bx=0.0;
        cx=2.0;
        tol = 1.e-8;

        v = brent(ax, bx, cx, f, tol, &t);                              //ブレント法による最小値を求める

        return t;
}
/*
        現在データとグリッドマップの位置合わせを行う


*/
int Get_Opt_Trans_4(void)
{
        int i,j,k;
        double ev,evx,evy,evc,ev_min;
        double kk;
        double xt,yt,ct;
        double dx,dy,dc;
        double dx1,dy1,dc1;
        double evthre,diff_ev,ev_old;
        double dd;
        int min_cnt;


        ev_min=999999999;

        dd              =0.00001;       //0.001
        evthre  =0.000001;      //0.001
        kk              =0.00001;       //0.0001

        min_cnt=0;

        xt=0;
        yt=0;
        ct=0;
        diff_ev=99999;
        OPT_XYC.x=xt;
        OPT_XYC.y=yt;
        OPT_XYC.c=ct;


//*****************初期の状態*********************************************************

        if(Kaiseki_Mode==0)
        ev=Get_Ev_Grid_ED(xt,yt,ct);                    //移動前のコスト関数

        else
        ev=Get_Ev_Grid_PD(xt,yt,ct);            //移動前のコスト関数

```

```
780             ev_min=ev;                                                                      //最低でもこれ以下
781
782     //****************最急降下法でコスト関数を最小化する*******************************
783             diff_ev=999999;
784             ev_old=ev;
785             k=0;
786             while(diff_ev>evthre){
787
788             if(Kaiseki_Mode==0)
789             evx=Get_Ev_Grid_ED(xt+dd,yt,ct);
790
790             else
791             evx=Get_Ev_Grid_PD(xt+dd,yt,ct);
792             dx1=(evx-ev)/dd;                                                //dx
793
794             if(Kaiseki_Mode==0)
795             evy=Get_Ev_Grid_ED(xt,yt+dd,ct);
796
796             else
797             evy=Get_Ev_Grid_PD(xt,yt+dd,ct);
798             dy1=(evy-ev)/dd;                                                //dy
799
800             if(Kaiseki_Mode==0)
801             evc=Get_Ev_Grid_ED(xt,yt,ct+dd);
802
802             else
803             evc=Get_Ev_Grid_PD(xt,yt,ct+dd);
804             dc1=(evc-ev)/dd;                                                //dc
805
806     //************アルミホ条件からステップ係数を求める*************************************************
807     //      kk=Get_alpha_in_Armijo(dx1,dy1,dc1,xt,yt,ct,ref_num);
808
809     //*************ブレント法からステップ係数を求める*************************************************
810             Gpose.x=xt;
811             Gpose.y=yt;
812             Gpose.c=ct;
813             Gdp.x=dx1;
814             Gdp.y=dy1;
815             Gdp.c=dc1;
816             kk=serch2();
817             kk=-kk;                                                 //符号を反転
818     //************値の更新*************************************************************************
819             xt=xt-kk*dx1;                                                                   //各パラメータの更新
820             yt=yt-kk*dy1;
821             ct=ct-kk*dc1;
822
823             if(Kaiseki_Mode==0)
824             ev=Get_Ev_Grid_ED(xt,yt,ct);
825             else
826             ev=Get_Ev_Grid_PD(xt,yt,ct);
827
828             if(ev<ev_min){
829             min_cnt=0;
830             ev_min=ev;
831
832             OPT_XYC.x=xt;
833             OPT_XYC.y=yt;
834             OPT_XYC.c=ct;
835             }
836
837             else{
838             min_cnt++;
839             }
840
841             diff_ev=ev_old-ev;                                                      //前回値の差分
842             ev_old=ev;                                                                      //前回値を更新
843             diff_ev=absF(diff_ev);
844
845             k++;
846             Iteration_Count=k;
847             if(k>100000)            break;
848             if(min_cnt>1000){
849             ev_min=-1;
850             break;
851             }
852
853             }
854             return ev_min;
855     }
856
857
858     float add_angle(float a1, float a2)
859     {
860       float sum = a1 + a2;
861       if (sum < -180)
862         sum += 360;
863       else if (sum >= 180)
864         sum -= 360;
865       return(sum);
866     }
867     /*
868             ICPによる地図作成のメインルーチン
869
870             マッチング誤差が大きい場合は地図を更新しない
871             スキャンの数が少なすぎる時も地図を更新しない
872
873             地図の更新に成功すると1を返す
874             地図の更新に失敗すると0を返す
875
876     */
877     int ICP_Grid(int idx)
878     {
879             int i,j,k,min_R,min_R_old,diff_R;
880             int flag;
881             float x,y,c,xg,yg;
882
883             flag=1;                                                                                 //フラグ=1
884
885             Measure_Num=SCAN_DATA_NUM[idx];                                                 //測定スキャンのデータ数
886
887             if(Measure_Num<100){                                                                    //測定スキャンのデータ数が100以下の場合
```

```
888             flag=0;                                                                                                 //解析失敗とみなす
889             ROBO_POS_2[idx]=ROBO_POS[idx];                                                          //ログ取り用変数に代入

890             }
891
892             else{
893     //*************参照SCAN点群を取得する***********************************
894             for(i=0;i<Measure_Num;i++){
895             Measure_scan[i]=Scan_Data[idx][i];                                                      //位置合わせするスキャンを呼び出す

896             }
897
898
899     //******************参照点の点間隔均一化*********************************
900             SCAN_DATA_NUM2[idx]=Resample_Point(SCAN_DATA_NUM[idx],dthreS);          //参照点の間隔を均一化する

901
902             Measure_Num=SCAN_DATA_NUM2[idx];                                                        //参照スキャンのデータ数

903             for(i=0;i<Measure_Num;i++){
904             Measure_scan[i]=Measure_scan2[i];                                                       //点間隔を均等化したスキャンをコピーする

905             }
906
907     //*************前回の移動量を反映させる*********************************

908
909     //*********************ICPのみ*****************************************************

910     /*
911             Trans_POS.x=0;
912             Trans_POS.y=0;
913             OPT_XYC_SUM.c=0;
914
915             for(i=0;i<OPT_XYC_NUM;i++){
916             Transform_OR(OPT_XYC_M[i].x,OPT_XYC_M[i].y,OPT_XYC_M[i].c);                     //有効?

917             OPT_XYC_SUM.c+=OPT_XYC_M[i].c;
918             }
919
920             ROBO_POS_G.x=Trans_POS.x;                                                               //ICPマッチングにより補正したオドメトリデータ

921             ROBO_POS_G.y=Trans_POS.y;                                                               //ICPマッチングにより補正したオドメトリデータ

922             ROBO_POS_G.c=OPT_XYC_SUM.c;
923
924             ROBO_POS_2[idx]=ROBO_POS_G;
925     */
926     //******************ICP+オドメトリ*************************************************

927             Trans_POS.x=0;
928             Trans_POS.y=0;
929             OPT_XYC_SUM.c=0;
930
931             ROBO_POS_G.x=ROBO_POS[idx].x;
932             ROBO_POS_G.y=ROBO_POS[idx].y;
933             ROBO_POS_G.c=ROBO_POS[idx].c-90;

934             Transform_OR(ROBO_POS_G.x,ROBO_POS_G.y,ROBO_POS_G.c);           //Measure_scan配列を移動(オドメトリに合わせる)

935
936             for(i=0;i<OPT_XYC_NUM;i++){
937             Transform_OR(OPT_XYC_M[i].x,OPT_XYC_M[i].y,OPT_XYC_M[i].c);     //Measure_scan配列を移動

938             OPT_XYC_SUM.c+=OPT_XYC_M[i].c;                                                                  //回転は総和を取る

939             }
940
941             ROBO_POS_G.x=Trans_POS.x;                                                               //ICPマッチングにより補正したオドメトリデータ

942             ROBO_POS_G.y=Trans_POS.y;                                                               //ICPマッチングにより補正したオドメトリデータ

943             ROBO_POS_G.c=OPT_XYC_SUM.c+ROBO_POS[idx].c-90;                          //ICPマッチングにより補正したオドメトリデータ

944
945             ROBO_POS_2[idx]=ROBO_POS_G;                                                             //ログ取り用変数に代入

946
947     //****************オドメトリのみ*************************************************

948     /*
949             Trans_POS.x=0;
950             Trans_POS.y=0;
951             OPT_XYC_SUM.c=0;
952
953             ROBO_POS_G.x=ROBO_POS[idx].x;
954             ROBO_POS_G.y=ROBO_POS[idx].y;
955             ROBO_POS_G.c=ROBO_POS[idx].c-90;

956             Transform_OR(ROBO_POS_G.x,ROBO_POS_G.y,ROBO_POS_G.c);

957
958             ROBO_POS_G.x=Trans_POS.x;                                                               //ICPマッチングにより補正したオドメトリデータ

959             ROBO_POS_G.y=Trans_POS.y;                                                               //ICPマッチングにより補正したオドメトリデータ

960             ROBO_POS_G.c=OPT_XYC_SUM.c+ROBO_POS[idx].c-90;

961
962             ROBO_POS_2[idx]=ROBO_POS_G;
963     */
964             min_R_old=0;
965
966             OPT_XYC_SUM.x=0;
967             OPT_XYC_SUM.y=0;
968             OPT_XYC_SUM.c=0;
969
970             xg=0;
971             yg=0;
972
973             for(j=0;j<Kaisu_MD;j++){
974     //*************対応点を求める*************************************************

975             Get_TaiouTen_Grid(Thred_D);                                                                     //Ref_Scanの対応点をMAP_DATAから探す

976
977     //*************コスト関数を最小化する*****************************************

978             min_R=Get_Opt_Trans_4();                                                                        //最急降下法により最小化したベクトル(OPT_XYC)を得る

979
980     //*************Measure_scan[]を最適値へ移動させる***********************************
```

```
981             Transform_OR(OPT_XYC.x,OPT_XYC.y,OPT_XYC.c);                                    //Measure_scan[]を移動させる

982
983             x=OPT_XYC.x;
984             y=OPT_XYC.y;
985             OPT_XYC_SUM.c=add_angle(OPT_XYC_SUM.c,OPT_XYC.c);                               //角度を加算し-180° ～+180に正規化する

986             c=OPT_XYC.c*PI/180.0;
987             xg=xg*cosf(c)-yg*sinf(c)+x;                                                             //座標を変換する

988             yg=xg*sinf(c)+yg*cosf(c)+y;                                                             //座標を変換する

989             OPT_XYC_SUM.x=xg;                                                                               //X,Y座標を積分
990             OPT_XYC_SUM.y=yg;                                                                               //X,Y座標を積分
991
992             diff_R=ABS(min_R-min_R_old);
993             min_R_old=min_R;
994             if(diff_R<1&&min_R<2000)                break;
995             }
996
997     //*************コスト関数の収束値が大きすぎる場合は地図を更新しない*******************************

998             if(min_R<10000){
999             OPT_XYC_M[OPT_XYC_NUM]=OPT_XYC_SUM;                                             //移動座標列を記録

1000            OPT_XYC_NUM++;                                                                                  //移動座標列を記録
1001            Get_Normal(Measure_Num,Measure_scan);                                   //位置合わせ完了した測定データの法線を求める

1002            Add_Points_to_GridMap_2();                                                              //位置合わせ済み測定データをグリッドマップに
登録
1003            }
1004            else
1005            flag=0;
1006            }//end of else
1007
1008            return flag;
1009    }
1010
1011
1012    /*
1013            ①ゲームパッドで操縦しスキャンデータを取得する

1014            ②地図データを作成する
1015            ③SDカードでPCとデータをやり取りする

1016            ④目的地まで走行
1017    */
1018    void Test_Auto_Drive_1(void)
1019    {
1020            int i,j,chk;
1021            int x,y;
1022            float c,d;
1023
1024    //      GamePad_Ctrl();                                                                 //ゲームパッドで適当に動かす(Lidarとオドメト
リデータを蓄積)
1025
1026            Init_Grid_Map();                                                                        //グリッドマップデータを初期化

1027
1028    //**************台車位置・姿勢の初期値*************************************************

1029            ROBO_POS_G.x=ROBO_POS[0].x;
1030            ROBO_POS_G.y=ROBO_POS[0].y;
1031            ROBO_POS_G.c=ROBO_POS[0].c-90.0;

1032            ROBO_POS_2[0]=ROBO_POS_G;                                                       //ログ取り用変数に代入

1033
1034    //*****************最初のスキャンを地図に登録*************************************

1035            for(i=0;i<SCAN_DATA_NUM[0];i++){

1036            Measure_scan[i]=Scan_Data[0][i];

1037            }
1038
1039            SCAN_DATA_NUM2[0]=Resample_Point(SCAN_DATA_NUM[0],dthreS);              //参照点の間隔を均一化する

1040            Measure_Num=SCAN_DATA_NUM2[0];
1041
1042            Get_Normal(Measure_Num,Measure_scan2);          //測定データの法線を求める

1043
1044            for(i=0;i<Measure_Num;i++){
1045            Measure_scan[i]=Measure_scan2[i];

1046            }
1047
1048            for(i=0;i<10;i++)                                                               //最初のスキャンはマップに多く取り入れる

1049            Add_Points_to_GridMap_2();                                              //点列のマップデータをグリッドマップに登録

1050    //*****************各スキャンでSLAMを実行し地図を構築***********************************

1051            OPT_XYC_NUM=0;
1052
1053            for(i=1;i<SCAN_NUM_ALL;i++){
1054            chk=ICP_Grid(i);
1055    //      if(chk==1)
1056    //      Pi();                                                                           //地図更新成功
1057    //      else
1058    //      PiPi();                                                                         //地図更新失敗
1059            }
1060    //*******************SLAMが終わった後の座標を記録しておく********************

1061            CAR_POS.x=ROBO_POS_2[SCAN_NUM_ALL-1].x;                 //SLAMが終わったときの位置X

1062            CAR_POS.y=ROBO_POS_2[SCAN_NUM_ALL-1].y;                 //SLAMが終わったときの位置Y

1063            CAR_POS.c=ROBO_POS_2[SCAN_NUM_ALL-1].c+90.0;    //SLAMが終わったときの姿勢(0のとき90deg)

1064
1065    /*
1066    //***************地図ファイル(SLAM後)を保存*****************************************

1067            Open_SD(0,FileName_GridMap);                            //Writeモードでオープン

1068            SD_Write_GridMap();                                                     //SDカードに保存

1069            Close_SD();                                                                     //SDカードをクローズ
1070    //***************ポジションファイルを保存*****************************

1071            Open_SD(0,FileName_RoboPos);                            //Writeモードでオープン
```

```
1072            SD_Write_RoboPos();                                             //SDカードに保存
1073            Close_SD();                                                     //SDカードをクローズ
1074
1075            PiPiPi();                                                       //終了の合図
1076            SW_Wait(4);                                                     //SW4入力待ち
1077
1078            Init_SD();                                                      //SDカードの抜き差しをしたので再度イニシャライズ
1079    //*************ゴール位置指定座標ファイルを読み込む*****************************
1080            Open_SD(1,FileName_GoalPos);                            //Readモードでオープン
1081            GOAL_POS=SD_Read_Zahyo();                                       //ゴール座標読み込み
1082            Close_SD();                                                     //SDカードをクローズ
1083
1084    //**************目的地までの軌道生成*******************************
1085            if(SP_T_NUM==1){
1086            Gen_Pass();                                                     //軌道生成
1087
1088    //**************地図ファイル(膨張処理後)を保存*****************************************
1089            Open_SD(0,FileName_PassMap);                            //Writeモードでオープン
1090            SD_Write_GridMap();                                             //SDカードに保存
1091            Close_SD();                                                     //SDカードをクローズ
1092    //**************ポジションファイル(スプライン補完)を保存**************************
1093            Open_SD(0,FileName_PassPos);                            //Writeモードでオープン
1094            SD_Write_G_Pos();                                               //SDカードに保存
1095            Close_SD();                                                     //SDカードをクローズ
1096            }
1097
1098    //**************スプライン補完のみ*****************************************************
1099            else{
1100    //      rs_out_data_int(SP_T_NUM);                                      //X座標
1101    //      rs_puts("¥n¥r");
1102    //      for(i=0;i<SP_T_NUM;i++){
1103    //      rs_out_data_int(SP_T_POS[i].x);                                 //X座標
1104    //      rs_out_data_int(SP_T_POS[i].y);                                 //X座標
1105    //      rs_puts("¥n¥r");
1106    //      }
1107            Gen_Pass_2();
1108    //**************地図ファイル(膨張処理後)を保存*****************************************
1109            Open_SD(0,FileName_PassMap);                            //Writeモードでオープン
1110            SD_Write_GridMap();                                             //SDカードに保存
1111            Close_SD();                                                     //SDカードをクローズ
1112    //**************ポジションファイル(スプライン補完)を保存**************************
1113            Open_SD(0,FileName_PassPos);                            //Writeモードでオープン
1114            SD_Write_G_Pos();                                               //SDカードに保存
1115            Close_SD();                                                     //SDカードをクローズ
1116            }
1117
1118
1119            PiPiPi();                                                       //終了の合図
1120            SW_Wait(4);                                                     //SW4入力待ち
1121
1122    //      rs_out_data_int(CAR_POS.x);                                     //X座標
1123    //      rs_out_data_int(CAR_POS.y);                                     //Y座標
1124    //      rs_out_data_int(CAR_POS.c);                                     //Y座標
1125    //      rs_puts("¥n¥r");
1126
1127    //      wait_msec(10000);
1128    //      PiPiPi();                                                       //
1129
1130            Traject_Move_2();                                               //軌道追従走行
1131
1132            SW_Wait(4);                                                     //SW4入力待ち
1133
1134    //      for(i=0;i<DEBUG_MAX;i++){
1135    //      out_data_rs_fixed_dot(DEBUG_F1[i]*1000.0,8,3);  //dc値のダンプ
1136    //      rs_puts("¥n¥r");
1137    //      }
1138
1139            Disp_POS_PID_Result();                                  //結果のダンプ
1140
1141            while(1){}                                                              //無限ループ
1142    */
1143
1144    }
```