

平成 29 年度

情報科学類

ソフトウェアサイエンス実験 B

テーマ「移動ロボットの行動プログラミング」

情報科学類 3 年      No.201500000

橘・シルフィンフォード

# 目次

第 1 章	実験全体を通して	1
1.1	実験概要 . . . . .	1
1.2	学んだこと・考察 . . . . .	1
1.3	謝辞 . . . . .	1
第 2 章	8 の字を走行	2
2.1	課題概要 . . . . .	2
2.2	解法 . . . . .	2
2.3	結果 . . . . .	2
2.4	考察 . . . . .	2
第 3 章	前方 1 m 以内に障害物を見つけたら停止	3
3.1	課題概要 . . . . .	3
3.2	解法 . . . . .	3
3.3	結果 . . . . .	3
3.4	考察 . . . . .	3
第 4 章	左側にある壁に沿って走らせる	4
4.1	課題概要 . . . . .	4
4.2	解法 . . . . .	4
4.3	結果 . . . . .	5
4.4	考察 . . . . .	5
第 5 章	座標変換したセンサデータをプロットする	6
5.1	課題概要 . . . . .	6
5.2	測域センサの概要 . . . . .	6
5.3	解法 . . . . .	6
5.4	結果 . . . . .	7
5.5	考察 . . . . .	7
第 6 章	ポールを周回しながら人が近づくと威嚇する	8
6.1	課題概要 . . . . .	8
6.2	解法 . . . . .	8

6.2.1	共通 . . . . .	8
6.2.2	ポールの探索 (state == Pole_setup) . . . . .	9
6.2.3	パトロール (state == Patrol) . . . . .	9
6.2.4	不審者発見 (state == Alien_found) . . . . .	10
6.2.5	不審者追従 (state == Alien_chase) . . . . .	10
6.2.6	不審者逃走 (state == Alien_lost) . . . . .	10
6.3	結果 . . . . .	10
6.4	考察 . . . . .	10
第 7 章	添付資料	11
7.1	8 の字走行 . . . . .	11
7.2	前方 1m 以内に障害物を見つけたら停止 . . . . .	13
7.3	左側にある壁に沿って走らせる兼柱の周りを周回する . . . . .	16
7.4	座標変換したセンサデータをプロットする . . . . .	19
7.5	ポールを周回しながら人が近づくと威嚇する . . . . .	23

# 第1章 実験全体を通して

## 1.1 実験概要

この実験は測域センサを搭載したロボットの動作をプログラミングして、要求される仕様（環境・条件）を満たす動作をロボットに行わせることを目的とする。ロボットは YP-Spur によって PC と通信を行い、プログラムからはライブラリを通じてアクセスする。具体的なプログラミング・実行においては、YP-Spur Coordinator がインストールされた Linux マシンを用いて C 言語プログラムから “*ypspur.h*” に記述された関数を呼び出すことでこれを実現する。以下に実行環境を示す。

PC	robozuki3 (ThinkPad X200)
カーネル	Linux 4.4.0-71
OS	Ubuntu13.10
YP-Spur	1.14.0

## 1.2 学んだこと・考察

ロボットのプログラムは現実の制約に縛られることが多く、座標を取得しても回転の多い動作をさせると値が狂ってしまうことなどには随分悩まされた。他にも、デバッグを行うにあたって Spur に覆い隠されている部分に入ってしまうと途端難易度が急上昇した。

しかし、おかげで新たなライブラリを使用することには慣れたと思う。最も重要なのは、まず仕様を確認してからプログラミングに取り組むことだ。特にロボットはライブラリが複雑で、現実世界を扱う必要があるので単位系も厳密に決まっている。これを確認せずしてプログラムは書けない。

座標変換や最終課題での円の方程式の導出などで線形代数で習得した知識を生かすことができた。単純なコードではあるが、知識をうまく活用しセンサと組み合わせて物体の中心推定まで行った。

C 言語のプログラムを書く力も上達した。隋分長いコードになってしまったのでライブラリ化することも考えたが、最終課題に必要な機能の実装が間に合わなかったのだからなかなかった。最終課題の重量は突然これまでと比較できないほど重い実装を行い非常に辛かったが、試行錯誤しながら自ら学び、また TA に教えてもらいながら自分にできるだけの力を出し切って挑戦できた。

## 1.3 謝辞

TA の空海さんからは課題に行き詰まったときに多くの助言をいただき設計に関する深い部分に関しても親身に評価・改善してくださりました。日蓮さんにはデバッグに協力していただいただけでなく、授業時間外にも実験について相談に乗っていただきました。ここに感謝の意を表します。

## 第2章 8の字を走行

これは講義資料「第2回 ロボットの動かし方と基本的な動作」内の課題です。

### 2.1 課題概要

ロボットを8の字を描くように走行させる。

なおこの実験ではセンサを利用しないため、周囲に何も無い場所で開始する必要がある。

### 2.2 解法

二つの円を描くにあたって円を切り替えるタイミングに注意する。先に反時計回りに上の円を描くのだが、原点のみで下の円の描画に切り替わるような判定をすると開始時に判定が行われ下の円しか描画されない。

そこで、右図のように  $(1,0)$ ,  $(0,0)$ ,  $(-1,0)$  の3点に達した（正確には3点の半径1cm以内に入った）タイミングで命令が切り替わるように設計した。以下に命令列を書き下した。

- $(0,1)$  に到達するまで上の円を描く
- $(0,0)$  に到達するまで上の円を描く
- $(0,-1)$  に到達するまで下の円を描く
- $(0,0)$  に到達するまで下の円を描く

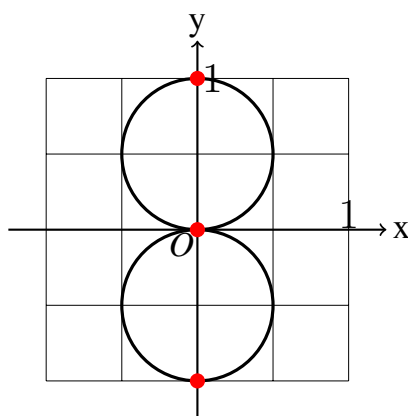


図 2.1 原点を出発して8の字を描くように進む

### 2.3 結果

右図のように綺麗なオドメトリが計測できた。

### 2.4 考察

単純な課題ではあるが、開始時に上の円の描画命令を発行した後プログラムをスリープさせる等切り替えタイミングの解決法は複数考えられる。

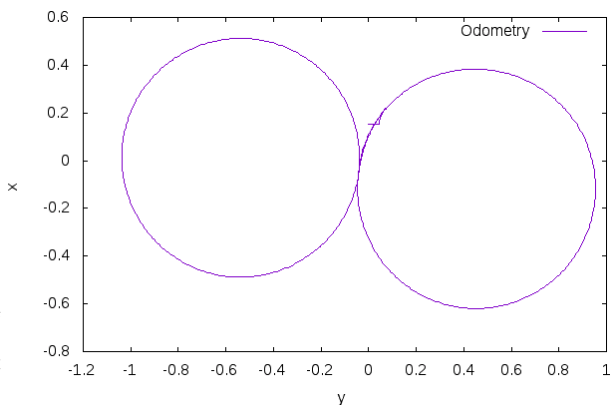


図 2.2 8の字走行のオドメトリ

## 第3章 前方1 m以内に障害物を見つけたら停止

これは講義資料「第3回 測域センサの使い方」の課題です。

### 3.1 課題概要

ロボットを直進させ、前方1 m以内に障害物を見つけたら停止する。

### 3.2 解法

センサから前方面の壁との距離を取得し、もし1m以下であれば速度を0に設定する。ただし、センサのスキャンデータ `scan->data` は単位がmmであることに注意し、1mのときは1000mm以下と指示する必要がある。

### 3.3 結果

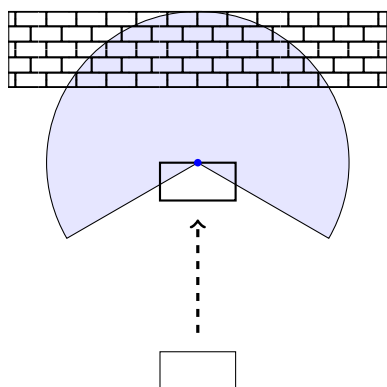


図 3.1 センサ視点



図 3.2 実験の様子

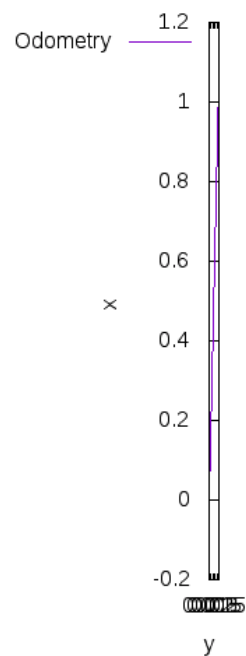


図 3.3 オドメトリ

### 3.4 考察

条件を切り分ける必要がなく課題としては単純だったが、単位に注意する良いきっかけになった。

## 第4章 左側にある壁に沿って走らせる

これは講義資料「第3回 測域センサの使い方」および中間課題1「所々に10cm以下の隙間がある約5mの壁に沿って走行」の課題です。

### 4.1 課題概要

鈍角のみで構成される壁に沿ってロボットを走らせる。中間課題では壁から何cm離れて走行するか制約が特に指定されていなかったのですが、第3回と同様に壁を左方50cmに見て走行するように設計を行った。また終了条件の制約もなかったため、壁の端まで行ったら回転して反対側の壁を走行するような無限ループを基本とする設計を行っている。

### 4.2 解法

資料のヒントにはセンサで読み取ったある2点の距離から傾きや位置を計算する手法が示されていたので、あえて1点の距離情報のみを使ってこれを解決できないか検討した。まず真左のセンサを利用することを考えたがこれはすぐに却下された。真左までの距離 $d$ は、ロボットが左右どちらに傾いても増加するからである。そこで図のように真左よりも $60^\circ$ 前方のセンサを利用することで、ロボットが左に傾けば $d$ が減少し、右に傾けば $d$ が増加するという理想の結果が得られた。

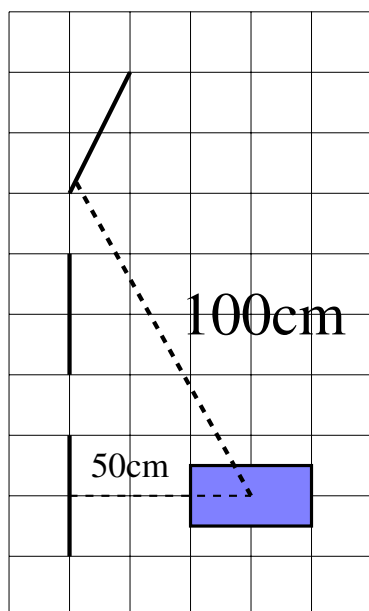


図 4.1 センサデータは左前方の一点を利用する

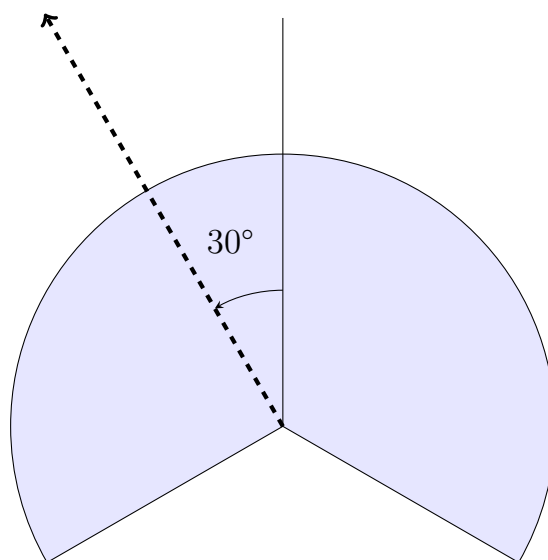


図 4.2 センサ範囲

左前方を見るというアイデアを使えば右図のような単純なアルゴリズムで壁に沿って進むことはできる。しかしこれではセンサが捕捉できない隙間に到達した時の挙動が不安定になる。

そこで過去のセンサ情報を保存するようにした。数回に一度それらをチェックして十分に長い時間壁を検出できていないようなら停止し、その場で左に大きく回転する。

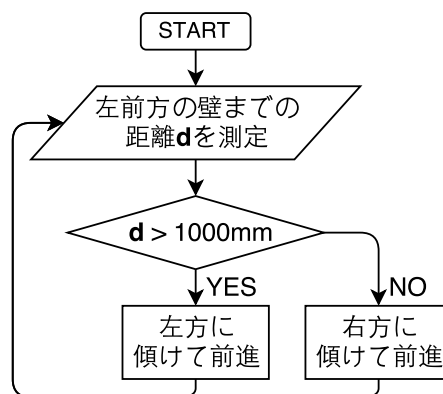


図 4.3 壁並走アルゴリズム案

## 4.3 結果

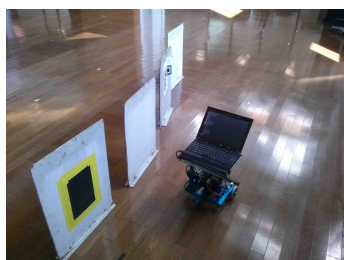


図 4.4 隙間のある壁

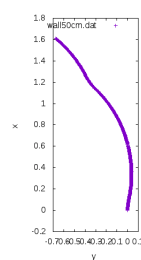


図 4.5 オドメトリ: 隙間のある壁

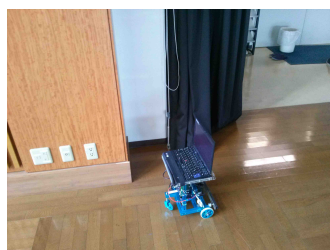


図 4.6 柱の周回

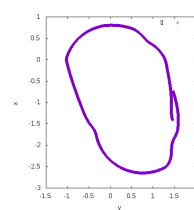


図 4.7 オドメトリ: 柱の周回

プログラムを書き換えることなく中間課題2「柱に沿って走行」の条件を満たした。

## 4.4 考察

センサ情報を1点しか使わないことにこだわったのは資料のヒントに逆らったことであったが、ここで考えた過去のセンサ情報を利用する手法が、結果的に第6章の最終課題で同一物体の座標を配列に保存して利用するアイデアの元となった。



## 第5章 座標変換したセンサデータをプロットする

これは講義資料「第4回 オドメトリと座標系」内の課題です。

### 5.1 課題概要

測域センサをつけたフリー状態（外力を受けて動く状態）のロボットを動かしてマッピングする。

ロボットを中心とした「方向」と「距離」の情報をもつ極座標系のデータをセンサから受け取り、ロボットを原点  $(0, 0)$ <sup>\*1</sup>とする直交座標系のデータ（FS 座標系データ）に変換するだけのプログラムが与えられる。これをスタート地点を原点  $(0, 0)$ とする平面に関して唯一の値をとる直交座標系のデータ（GL 座標系データ）に変換する処理を記述してマッピングを行う。

### 5.2 測域センサの概要

測域センサには北陽電機株式会社の「URG-04LX」を使用する。ロボットの前方に取り付けられたセンサが PC と USB 接続され、プロトコル SCIP2.0 を利用して通信を行う。

### 5.3 解法

FS 座標系も GL 座標系も同一平面上の点の位置を定めるものなので、FS 座標系で表現される点を GL 座標系に変換するには、現在地の GL 座標とスタート地点において  $\theta = 0$  と定義するロボットの傾き  $\theta(\text{rad})$  が必要となる。

具体的な処理を明確にするためにプログラムを設計する前に右のような図を描いた。目的は、FS 座標  $(Px, Py)$  で表現される赤い点を GL 座標  $(glPx, glPy)$  に変換することだ。右図において XY 軸が一般的な図からは 90 度ほど回転しているが、直進すると X 座標の値が増え、左に進むと Y 座標の値が増える仕様に合わせたためだ。

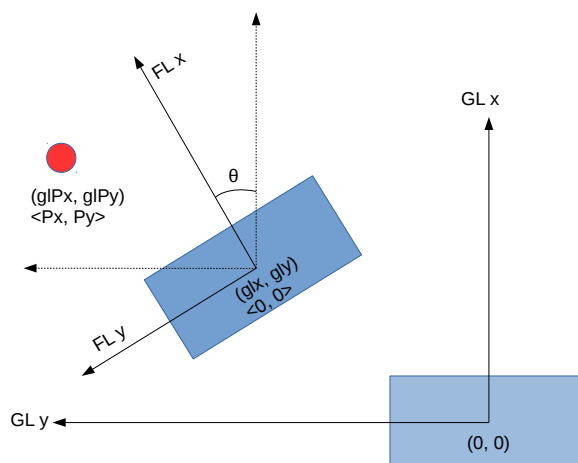


図 5.1 FS 座標系と GL 座標系の関係

<sup>\*1</sup> (X 座標, Y 座標)

まず現在のロボットの傾き  $pos\_theta\_gl$  を用いて物体の座標を回転させる。座標の回転といっても同一平面上の点の回転移動と同じであるから、以下のような行列計算で求めることができる。

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(pos\_theta\_gl) & -\sin(pos\_theta\_gl) \\ \sin(pos\_theta\_gl) & \cos(pos\_theta\_gl) \end{pmatrix} \begin{pmatrix} x\_fs \\ y\_fs \end{pmatrix}$$

上記の記述でロボットの傾き  $\theta$  が 0 のときと同じ座標系に移すことができたが、これでは原点がロボット中心になりマップを作ることができない。ここで現在地の GL 座標 ( $pos\_x\_gl, pos\_y\_gl$ ) を加算する。これは GL 座標の原点から FS 座標の原点（ロボット中心）への単純な平行移動で実現される。先の行列計算に合わせて記述すると以下ようになる。

$$\begin{pmatrix} x\_gl \\ y\_gl \end{pmatrix} = \begin{pmatrix} pos\_x\_gl \\ pos\_y\_gl \end{pmatrix} + \begin{pmatrix} \cos(pos\_theta\_gl) & -\sin(pos\_theta\_gl) \\ \sin(pos\_theta\_gl) & \cos(pos\_theta\_gl) \end{pmatrix} \begin{pmatrix} x\_fs \\ y\_fs \end{pmatrix}$$

こうして得られる物体の座標は GL 座標の原点を基準とした平面で唯一の座標である。

## 5.4 結果

実際にフリー状態のロボットを動かしてマッピングしてみる。

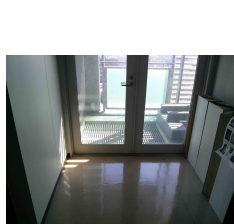


図 5.2 実際の場所 A

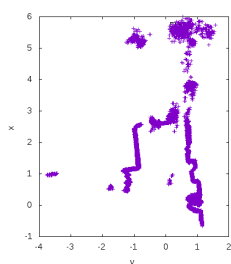


図 5.3 作成したマップ A



図 5.4 実際の場所 B

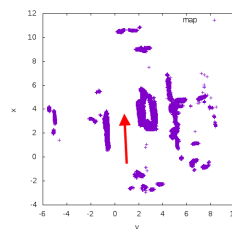


図 5.5 作成したマップ B  
( $y=6$  の壁は即席のもの)

ノイズが目立つが正しくマッピングできている。

## 5.5 考察

座標系の変換がセンサをより強力なツールに変えた。URG から送られるデータはロボットから壁（単に光を反射するもの）までのベクトルに過ぎないが、ロボットの位置情報を使った座標系の変換を行うことで、世界中の壁を記録することができる。

第 6 章の最終課題では、これをさらに物体認識を行うツールへと強化する。これにより単なる壁の記録のみならず、同一物体を検出しオブジェクトとして記録できるようになる。

## 第6章 ポールを周回しながら人が近づくと威嚇する

これは最終課題5「警備員」の課題です。

### 6.1 課題概要

ロボットはポールを中心とした半径 50cm の円を反時計回りに周回する。ポールの直径は約 12cm。初期位置の制約は定められていないので、前方もしくは左方にポールを見るような地点ならば任意の場所からスタート可能なように設計を行った。周回中にセンサが人に反応した場合、人とポールの間立ち威嚇を行う。威嚇時の動作は定められていなかったため、ポールから 2m 以内の範囲で追いかける設計とした。センサが人を捉えられなくなったり、ポールから 2m 以上離れてしまった場合は再びポールの周回に戻る。

### 6.2 解法

物体認識やそれに伴う時間に依存しない（つまり次のサイクルに持ち越される）同一物体判定など複雑な処理が多く、また要求される動作の種類もある程度数が予想できたので、現在の状態を表す変数 *robostate* を作成し、ロボットの動作が以下の図のように状態遷移していくような設計にした。

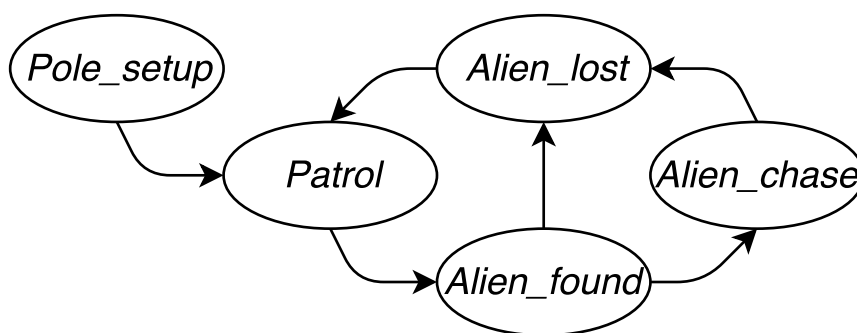


図 6.1 作成したプログラムの状態遷移図

#### 6.2.1 共通

第5章と同じ方法で物体の座標を計算し、十分近くの座標同士を同一の点群に属するものとしてオブジェクト化する。ここで、OBJECT 構造体（単なる座標データ）の配列をオブジェクトと呼称することで若干の抽象化を図る。同一の物体と判定を受けたものは同一の配列（OBJECT 構造体の配列）に挿入され、同一でない物体はまた別の配列に挿入される。同一物体の座標は定数 *OBJBUF* を超えない範囲で配列に挿入される。


OBJECT [0]	[1]	[2]	[3]	...	[OBJBUF-1]
(x, y)					...
use					...

図 6.2 オブジェクトの実体

今の実装では配列の最初 (*object[0]*) は、その配列にどこまで使用可能な値が入っているかを示す変数 *use* を使用するためだけに使用しているので座標は入れない。それ以外の *use* は使用していれば 1、使用していなければ 0 が入っている。

同一物体判定アルゴリズムの解説を行うとレポートのページ制限を軽く超えてしまうので、詳しくは添付資料内の関数 *pigeonhole* を参照していただきたい。

## 6.2.2 ポールの探索 (state == Pole\_setup)

まずオブジェクトの中で最も多く座標が取得されているものをポールの候補とする。ポール候補となったオブジェクトの任意の 3 点から円の方程式を計算する (関数 *p3\_to\_circle* 参照)。

求めた円の方程式の半径 *r* が実際のポールの半径と矛盾しなければ、ポールの中心を LC 座標系の原点に設定して変数 *state* を *Patrol* に移行する。矛盾する場合間違ったオブジェクトを認識したと判断し、保存されているオブジェクトを全てリセットして最初からやり直す。

ポールの探索段階では人や壁などのセンサに反応するポール以外の物体が周りにないことを仮定している。ただセンサは前方から左側のみを使用するので、前方もしくは左方にポールを見据える形であれば右方に物体があっても動作する。

## 6.2.3 パトロール (state == Patrol)

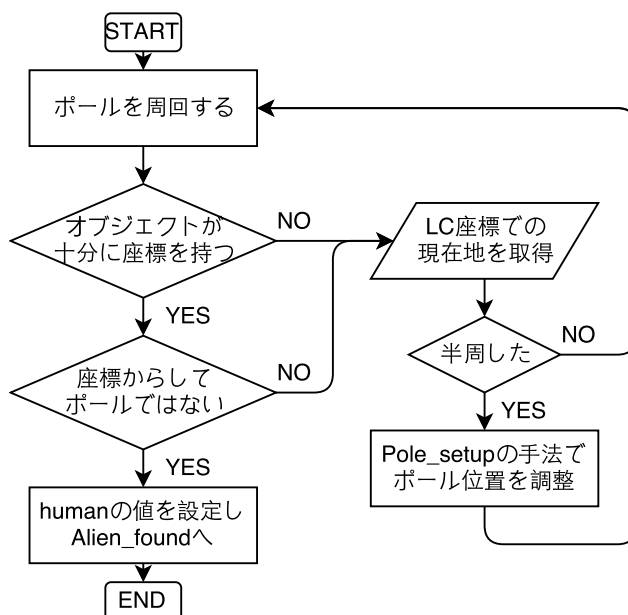


図 6.3 パトロール中の動作チャートフロー

細かいループ等実装上の細かいポイントを詳細に記述するとあまりに巨大になるので説明上問題にならない程度に省略している。実際には本体のソースコードを読んでもらった方が良い。

特に重要な点としては、このプログラムでは *Pole\_setup* 以外の状態では LC 座標を基準に動いているということだ。フローチャート内に書いた「*Pole\_setup* の手法」はその通りなのだが、基準が GL 座標から LC 座標に変わっていること。また、ポール位置の座標を座標系の中で修正するのではなく、ポール位置から座標系を修正するという挙動が極めて重要である。あくまでポールの中心が LC 座標の原点となるように設計した。

## 6.2.4 不審者発見 (state == Alien\_found)

*Patrol* で得た human の値を使ってロボットと不審者の間に割り込んでいく。

## 6.2.5 不審者追従 (state == Alien\_chase)

ロボットが不審者を見失うことなくポールと不審者の間に入ると不審者の追従が始まる。不審者を見失うまでの間、ロボットが前方に不審者を見据えた状態で数歩後ろをついて行く。

## 6.2.6 不審者逃走 (state == Alien\_lost)

ロボットが不審者を見失ったと判断する条件は 2 つある。周りにオブジェクトが 1 つもなくなったときと、ポールとの距離が定数 *LOSTAREA* を超えた（現在設定はポールから 2m 離れた）ときだ。

不審者を見失うと周回経路に戻り、ポールの周り 50cm に入ると変数 *state* を *Patrol* に移行する。

## 6.3 結果

不審者追従までは最終発表後に動作させることができたが、不審者逃走後は実装が間に合わず、ポールを不審者と誤認識してしまう。以下は動作中の様子例である。

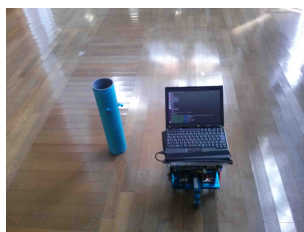


図 6.4 周回中のロボット



図 6.5 追跡中のロボット

## 6.4 考察

時間内に完成させられなかったのは悔しいが、これまでの課題で学んだことの集大成となる課題であり非常に楽しかった。特にセンサノイズ除去に力を入れたので添付資料を見て欲しい。

## 第7章 添付資料

### 7.1 8の字走行

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <math.h>
4  #include <yypspur.h>
5
6  void iamat(void);
7
8  int main( void )
9  {
10     double x, y, theta;
11
12     if ( Spur_init() < 0 )
13     {
14         fprintf(stderr, "ERROR : cannot open spur.\n");
15         return -1;
16     }
17
18     Spur_set_pos_GL( 0, 0, 0 );
19     Spur_set_pos_LC( 0, 0, 0 );
20
21     // ちょっと加速
22     Spur_set_vel( 0.2 );
23     Spur_set_accel( 1.0 );
24     Spur_set_angvel( M_PI );
25     Spur_set_angaccel( M_PI );
26
27     Spur_circle_GL( 0, 0.5, 0.5 );
28     while( !Spur_near_pos_GL( 0, 1.0, 0.01 ) ) {
29         iamat();
30         usleep( 10000 );
31     }
32
33     Spur_circle_GL( 0, 0.5, 0.5 );
34     while( !Spur_near_pos_GL( 0, 0, 0.01 ) ) {
35         iamat();
36         usleep( 10000 );
37     }
38
39     Spur_circle_GL( 0, -0.5, -0.5 );
40     while( !Spur_near_pos_GL( 0, -1.0, 0.01 ) ) {
41         iamat();
42         usleep( 10000 );
43     }
44
45     Spur_circle_GL( 0, -0.5, -0.5 );
46     while( !Spur_near_pos_GL( 0, 0, 0.01 ) ) {
```

```

47     iamat();
48     usleep( 10000 );
49 }
50
51 Spur_stop( );
52 usleep( 40000 );
53 Spur_free( );
54 printf( "Hit Ctrl-C to exit.\n" );
55 while( 1 )
56 {
57     Spur_get_pos_GL( &x, &y, &theta );
58     printf( "%f %f %f\n", x, y, theta * 180.0 / M_PI );
59     usleep( 1000000 );
60 }
61
62 return 0;
63 }
64
65 void iamat(void)
66 {
67     double real_x, real_y, real_th;
68     Spur_get_pos_GL( &real_x, &real_y, &real_th );
69     printf("(%f, %f)\n", real_x, real_y);
70 }

```

Listing 7.1 8 の字走行

## 7.2 前方 1m 以内に障害物を見つけたら停止

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/time.h>
4  #include <signal.h>
5  #include <math.h>
6  #include <yypur.h>
7  #include <scip2awd.h>
8
9  int escape;
10
11 void ctrlc( int notused )
12 {
13     // MDコマンドを発行したままプログラムを終了すると、
14     // 次回起動時に少し余分に時間がかかる
15     escape = 1;
16     signal( SIGINT, NULL );
17 }
18
19 int main( int argc, char *argv[] )
20 {
21     S2Port *port;    // ポート
22     S2Sdd_t buf;     // データ取得用ダブルバッファ
23     S2Scan_t *scan;  // データ読み出し用構造体
24     S2Param_t param; // センサのパラメータ構造体
25
26     int ret;
27
28     if( argc != 2 ){
29         fprintf( stderr, "USAGE: %s device\n", argv[0] );
30         return 0;
31     }
32
33     if ( Spur_init() < 0 )
34     {
35         fprintf(stderr, "ERROR : cannot open spur.\n");
36
37         return -1;
38     }
39
40     Spur_set_pos_GL( 0, 0, 0 );
41     Spur_set_pos_LC( 0, 0, 0 );
42
43     Spur_set_vel( 0.2 );
44     Spur_set_accel( 1.0 );
45     Spur_set_angvel( M_PI );
46     Spur_set_angaccel( M_PI );
47
48     //Spur_stop_line_GL( 5.0, 0.0, M_PI / 2 );
49     Spur_stop_line_GL( 5.0, 0.0, 0.0 );
50
51     // ポートを開く
52     port = Scip2_Open( argv[1], B0 );
```



```

53     if( port == 0 ){
54         fprintf( stderr, "ERROR: Failed to open device.\n" );
55         return 0;
56     }
57     printf( "Port opened\n" );
58
59     // 初期化
60     escape = 0;
61     signal( SIGINT, ctrlc );
62     S2Sdd_Init( &buf );
63     printf( "Buffer initialized\n" );
64
65     // URGのパラメータ取得
66     Scip2CMD_PP( port, &param );
67
68     // URG-04LXの全方向のデータを取得開始
69     Scip2CMD_StartMS( port, param.step_min, param.step_max,
70         1, 0, 0, &buf, SCIP2_ENC_3BYTE );
71
72     while( !escape ){
73         ret = S2Sdd_Begin( &buf, &scan );
74         if( ret > 0 ){
75             // 新しいデータがあった時の処理をここで行う
76             printf( "Front distance: %lu mm\n",
77                 scan->data[ param.step_front - param.step_min ] );
78             if ( scan->data[ param.step_front - param.step_min ] < 1000.0 )
79             {
80                 Spur_set_vel( 0.0 );
81                 Spur_set_accel( -2.5 );
82                 escape = 1;
83             }
84             // S2Sdd_BeginとS2Sdd_Endの間でのみ、構造体scanの中身にアクセス可能
85             S2Sdd_End( &buf );
86         }
87         else if( ret == -1 ){
88             // 致命的なエラー時(URGのケーブルが外れたときなど)
89             fprintf( stderr, "ERROR: Fatal error occurred.\n" );
90             break;
91         }
92         else{
93             // 新しいデータはまだ無い
94             usleep( 10000 );
95         }
96     }
97     printf( "\nStopping\n" );
98
99     ret = Scip2CMD_StopMS( port, &buf );
100    if( ret == 0 ){
101        fprintf( stderr, "ERROR: StopMS failed.\n" );
102        return 0;
103    }
104
105    printf( "Stopped\n" );
106    S2Sdd_Dest( &buf );
107    printf( "Buffer destructed\n" );
108    Scip2_Close( port );

```

```
109     printf( "Port closed\n" );
110
111     return 1;
112 }
```

Listing 7.2 前方 1m 以内に障害物を見つけたら停止

### 7.3 左側にある壁に沿って走らせる兼柱の周りを周回する

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/time.h>
4  #include <signal.h>
5  #include <math.h>
6  #include <ypspur.h>
7  #include <scip2awd.h>
8
9  #define GOAL_DISTANCE    500
10 #define GOAL_DIAGONAL    GOAL_DISTANCE*2
11 #define GOAL_ACCURACY    10.0
12 #define GOAL_MIN         GOAL_DIAGONAL-GOAL_ACCURACY
13 #define GOAL_MAX         GOAL_DIAGONAL+GOAL_ACCURACY
14 #define CHECKTIME       10
15
16 int  escape;
17 float prediag_data[CHECKTIME];
18
19 void ctrlc(int notused)
20 {
21     escape = 1;
22     signal(SIGINT, NULL);
23 }
24
25 int main(int argc, char *argv[])
26 {
27     S2Port *port;    // ポート
28     S2Sdd_t buf;     // データ取得用ダブルバッファ
29     S2Scan_t *scan;  // データ読み出し用構造体
30     S2Param_t param; // センサのパラメータ構造体
31
32     int cycle, ret;
33
34     if (argc != 2) {
35         fprintf(stderr, "USAGE: %s device\n", argv[0]);
36         return 0;
37     }
38
39     if (Spur_init() < 0)
40     {
41         fprintf(stderr, "ERROR : cannot open spur.\n");
42
43         return -1;
44     }
45
46     Spur_set_pos_GL(0, 0, 0);
47     Spur_set_pos_LC(0, 0, 0);
48
49     Spur_set_vel(0.1);
50     Spur_set_accel(0.5);
51     Spur_set_angvel(M_PI/4);
52     Spur_set_angaccel(M_PI);
```

```

53
54 // ポートを開く
55 port = Scip2_Open(argv[1], B0);
56 if (port == 0){
57     fprintf(stderr, "ERROR: Failed to open device.\n");
58     return 0;
59 }
60 printf("Port opened\n");
61
62 // 初期化
63 cycle = 1;
64 escape = 0;
65 prediag_data[0] = 500.0;
66 signal(SIGINT, ctrlc);
67 S2Sdd_Init(&buf);
68 printf("Buffer initialized\n");
69
70 // URGのパラメータ取得
71 Scip2CMD_PP(port, &param);
72
73 // URG-04LXの全方向のデータを取得開始
74 Scip2CMD_StartMS(port, param.step_min, param.step_max,
75     1, 0, 0, &buf, SCIP2_ENC_3BYTE);
76
77 while (!escape) {
78     ret = S2Sdd_Begin(&buf, &scan);
79     if (ret > 0){
80         int nowall = 0;
81         unsigned long diag_data;
82
83         diag_data = scan->data[param.step_front - param.step_min
84             + param.step_resolution/12];
85
86         // 定期的に壁の横にいるか検出する
87         if (cycle == CHECKTIME-1) {
88             while (cycle != 1) {
89                 // 壁の横にいないなら左前方に壁が見えるまでその場で回転する
90                 if (prediag_data[cycle] < 20.0 || 2000.0 < prediag_data[cycle])
91                     nowall++;
92                 cycle--;
93                 printf("%d(%ld) : %d\n", cycle, diag_data, nowall);
94             }
95             if (CHECKTIME/4 < nowall) {
96                 printf("Adjust\n");
97                 Spur_spin_FS(M_PI/16);
98                 sleep(1);
99             } else {
100                 prediag_data[0] = prediag_data[CHECKTIME-1];
101                 cycle = 1;
102             }
103             nowall = 0;
104         }
105
106         printf ("distance to wall is %ld\n", diag_data);
107
108         Spur_orient_FS(0);

```

```

109
110     if (GOAL_MIN <= diag_data && diag_data <= GOAL_MAX)
111         Spur_orient_FS((prediag_data[cycle-1] < diag_data) ?
112             M_PI/256 : -M_PI/256);
113     else
114         Spur_orient_FS((GOAL_MAX < diag_data) ?
115             M_PI/128 : -M_PI/128);
116
117     prediag_data[cycle] = diag_data;
118     cycle++;
119
120     S2Sdd_End(&buf);
121 }
122 else if (ret == -1){
123     // 致命的なエラー時(URGのケーブルが外れたときなど)
124     fprintf(stderr, "ERROR: Fatal error occurred.\n");
125     break;
126 }
127 else{
128     // 新しいデータはまだ無い
129     usleep(100000);
130 }
131 }
132
133 Spur_set_vel(0.0);
134 Spur_set_accel(-1.0);
135 printf("\nStopping\n");
136
137 ret = Scip2CMD_StopMS(port, &buf);
138 if (ret == 0){
139     fprintf(stderr, "ERROR: StopMS failed.\n");
140     return 0;
141 }
142
143 printf("Stopped\n");
144 S2Sdd_Dest(&buf);
145 printf("Buffer destructed\n");
146 Scip2_Close(port);
147 printf("Port closed\n");
148
149 return 1;
150 }

```

Listing 7.3 左側にある壁に沿って走らせる

## 7.4 座標変換したセンサデータをプロットする

```
1  /*
2  *   ロボット-センサ座標系変換サンプルプログラム
3  *   Original: 2011/04/19 Taku Shikina
4  *   Modified: 2012/12/10 Atsushi Watanabe
5  */
6
7  /*-----*/
8  // ライブラリ
9  /*-----*/
10 #include <stdio.h>                // printfなどの入出力関数
11 #include <stdlib.h>               // mallocとfreeやexitなど
12 #include <math.h>                 // 数学関数
13 #include <signal.h>               // シグナル
14
15 #include <yypspur.h>               // spur用ライブラリ
16 #include <scip2awd.h>             // AWDのURGライブラリ
17
18 int g_escape;
19
20 /*=====*/
21 // 関数名      : ctrlc
22 // 機能概要    : シグナルハンドラの設定. Ctrl+Cでプログラムを終了させる
23 /*=====*/
24 void ctrlC( int aStatus )
25 {
26     signal( SIGINT, NULL );
27     g_escape = 1;                // 終了予約
28 }
29
30 void setctrlC( )
31 {
32     signal( SIGINT, ctrlC );
33 }
34
35 /*-----*/
36 // メインプログラム
37 /*-----*/
38 int main( int aArgc, char **aArgv )
39 {
40     int ret;
41     int i;
42
43     S2Port *urgPort;              // デバイスファイル名
44     S2Sdd_t urgBuf;               // データを確保するバッファ
45     S2Scan_t *urgData;            // バッファへのポインタ
46     S2Param_t urgParam;           // URGのパラメータを確保
47
48     if( aArgc < 2 )
49     {
50         fprintf( stderr, "USAGE: %s /dev/ttyACM0\n", aArgv[0] );
51         return 0;
52     }
```

```

53 g_escape = 0;
54 setctrlC( ); // シグナルハンドラの設定
55
56 /*****spur関連 *****/
57 ret = Spur_init( ); // 初期化
58 if( ret <= 0 )
59 {
60     fprintf( stderr, "ERROR: Failed to open yp-spur.\n" );
61     return 0;
62 }
63 Spur_set_vel( 0.15 ); // 最大速度0.15m/sに設定
64 Spur_set_accel( 1.0 ); // 加速度1.0m/sに設定
65 Spur_set_angvel( 0.25 ); // 最大角速度0.25rad/sに設定
66 Spur_set_angaccel( 2.0 ); // 各加速度2.0rad/ssに設定
67 Spur_set_pos_GL( 0, 0, 0 ); // スタート地点を原点にGL座標を設定
68
69 /*****
70
71 /*****URG関連 *****/
72 /* ポートオープン */
73 urgPort = Scip2_Open( aArgv[1], B0 ); // デバイス名,ボーレート設定
74
75 if( urgPort == 0 )
76 {
77     fprintf( stderr, "ERROR: Failed to open device. %s\n", aArgv[1] );
78     return 0;
79 }
80 fprintf( stderr, "Port opened\n" );
81
82 /* バッファの初期化 */
83 S2Sdd_Init( &urgBuf );
84 fprintf( stderr, "Buffer initialized\n" );
85
86 /* URGパラメータの読み出し */
87 Scip2CMD_PP( urgPort, &urgParam );
88
89 /* 垂れ流しモードの開始 */
90 Scip2CMD_StartMS( urgPort, urgParam.step_min, urgParam.step_max, 1, 0, 0,
91                  &urgBuf, SCIP2_ENC_3BYTE );
92
93 /* 定数の計算 */
94 double resolution = 2.0 * M_PI / urgParam.step_resolution;
95
96 /*****
97
98 /*****ロボットの座標を考慮したURGデータの使用 *****/
99
100 Spur_free( );
101
102 while( !g_escape )
103 {
104
105     /* 測位データの取り出し */
106     ret = S2Sdd_Begin( &urgBuf, &urgData );
107
108     if( ret > 0 )

```

```

109 {
110     double d, theta; // (mm), (rad) URGの生データ (極座標系)
111     double x_sensor, y_sensor; // (m) URGのデータ (センサ座標系)
112     double x_fs, y_fs; // (m) FS座標系のURGのデータ (ロボット座標系)
113     double x_gl, y_gl; // (m) GL座標系のURGのデータ (世界座標系)
114     double pos_x_gl, pos_y_gl, pos_theta_gl; // (m), (rad) GL座標系のロボットの位置
115
116     // GL座標系のロボットの位置 (世界座標系)
117
118     // ロボットの現在位置を取得
119     Spur_get_pos_GL( &pos_x_gl, &pos_y_gl, &pos_theta_gl );
120     //printf( "X: %f\tY: %f\tth: %f\n", pos_x_gl, pos_y_gl, pos_theta_gl );
121
122     // センサデータをGL座標系に張り付けて出力
123     for ( i = 0; i < urgData->size; i++ )
124     {
125         // 極座標系のデータの取得
126         d = urgData->data[i];
127         theta = ( double )( urgParam.step_min + i - urgParam.step_front )
128                 * resolution;
129
130         if( d < urgParam.dist_min || d > urgParam.dist_max )
131             continue;
132
133         // 極座標系からセンサ座標系への変換
134         x_sensor = d * cos( theta ) * 0.001; // mm -> m
135         y_sensor = d * sin( theta ) * 0.001; // mm -> m
136
137         // センサ座標系からFS座標系への変換
138         x_fs = x_sensor;
139         y_fs = y_sensor;
140
141         // FS座標系からGL座標系への変換
142         x_gl = pos_x_gl + x_fs * cos( pos_theta_gl ) - y_fs * sin( pos_theta_gl );
143         y_gl = pos_y_gl + y_fs * cos( pos_theta_gl ) + x_fs * sin( pos_theta_gl );
144
145         // 出力
146         printf( "%f\t%f\n", x_gl, y_gl );
147     }
148
149     S2Sdd_End( &urgBuf ); // アンロック (読み込み終了)
150
151 }
152 else if( ret < 0 )
153 {
154     // 戻り値が負だとエラー
155     fprintf( stderr, "ERROR: Fatal error occurred.\n" );
156     break;
157 }
158 else
159 {
160     usleep( 10000 ); // 測域データに新しいデータがない
161 }
162
163 }
164

```



```

165  /*****
166
167  /*****終了処理 *****/
168
169  // YP-Spurの停止
170  Spur_stop( );
171
172  // URGの終了
173  ret = Scip2CMD_StopMS( urgPort, &urgBuf );    // URGの測域停止
174  if( ret == 0 )
175  {
176      fprintf( stderr, "ERROR: StopMS failed.\n" );
177      return 0;
178  }
179  fprintf( stderr, "Stopped\n" );
180
181  S2Sdd_Dest( &urgBuf );                        // バッファの解放
182  fprintf( stderr, "Buffer destructed\n" );
183
184  Scip2_Close( urgPort );                       // ポートを閉じる
185  fprintf( stderr, "Port closed\n" );
186
187  return 0;
188 }

```

Listing 7.4 座標変換したセンサデータをプロットする

## 7.5 ポールを周回しながら人が近づくと威嚇する

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/time.h>
4  #include <signal.h>
5  #include <math.h>
6  #include <string.h>
7  #include <yypur.h>
8  #include <scip2awd.h>
9
10 // Spurに投げる命令の単位
11 #define POLE_R      0.1
12 #define DISTANCE    0.5
13 #define LOSTAREA    2
14
15 #define MAXOBJ      8
16 #define OBJBUF      512
17
18 enum robostate {
19     Pole_setup,
20     Patrol,
21     Alien_found,
22     Alien_chase,
23     Alien_lost
24 };
25
26 typedef struct {
27     int use;
28     double x, y;
29 } OBJECT;
30
31 enum robostate state = Pole_setup;
32 OBJECT obj[MAXOBJ][OBJBUF];
33 double human_x, human_y, human_t;
34 int prev_use[MAXOBJ];
35 int escape;
36
37 void ctrlc(int notused)
38 {
39     escape = 1;
40     signal(SIGINT, NULL);
41 }
42
43 void gl_to_lc(double glx, double gly, double *lcx, double *lcy)
44 {
45     double pos_x_gl, pos_y_gl, pos_theta_gl;
46     double pos_x_lc, pos_y_lc, pos_theta_lc;
47
48     Spur_get_pos_GL(&pos_x_gl, &pos_y_gl, &pos_theta_gl);
49     Spur_get_pos_LC(&pos_x_lc, &pos_y_lc, &pos_theta_lc);
50
51     *lcx = glx + pos_x_lc - pos_x_gl;
52     *lcy = gly + pos_y_lc - pos_y_gl;
```

```

53
54     return;
55 }
56
57 // 物体を覆う円の半径を求めるのに使いたいが、
58 // 物体の中心は円でないと求められない
59 int within_range(OBJECT p, OBJECT q, double range)
60 {
61     //printf("%fm %fm", hypot((p.x-q.x), (p.y-q.y)), range);
62     return (hypot((p.x-q.x), (p.y-q.y)) <= range) ? 1 : 0;
63 }
64
65 /* 参考: http://www.iot-kyoto.com/sato/2016/01/29/tangent-003 */
66 void p3_to_circle(OBJECT *o, double *x, double *y, double *r)
67 {
68     int p1, p2, p3;
69     double a, b, c, d;
70     double x1, y1, x2, y2, x3, y3;
71
72     p1 = o[0].use;
73     p2 = p1*2/3;
74     p3 = p1/3;
75
76     x1 = o[p1].x;
77     y1 = o[p1].y;
78     x2 = o[p2].x;
79     y2 = o[p2].y;
80     x3 = o[p3].x;
81     y3 = o[p3].y;
82
83     a = x2 - x1;
84     b = y2 - y1;
85     c = x3 - x1;
86     d = y3 - y1;
87
88     *x = x1 + (d*(a*a + b*b) - b*(c*c + d*d)) / (a*d - b*c) / 2 ;
89
90     *y = b ? (a*(x1+x2-(*x))-(*x)) + b*(y1+y2)) / b / 2
91           : (c*(x1+x3-(*x))-(*x)) + d*(y1+y3)) / d / 2;
92
93     *r = (sqrt(((x)-x1) * ((x)-x1) + ((y)-y1) * ((y)-y1)) +
94           sqrt(((x)-x2) * ((x)-x2) + ((y)-y2) * ((y)-y2)) +
95           sqrt(((x)-x3) * ((x)-x3) + ((y)-y3) * ((y)-y3))) / 3;
96
97     return;
98 }
99
100 void discard_obj(int objnum)
101 {
102     int i;
103     for (i = 0; i < OBJBUF; i++)
104         obj[objnum][i].use = obj[objnum][i].x = obj[objnum][i].y = 0;
105     return;
106 }
107
108 int mostuse() {

```

```

109     int i, m = 0;
110
111     for (i = 0; i < MAXOBJ; i++)
112         m = (m > obj[i][0].use) ? m : obj[i][0].use;
113
114     for (i = 0; i < MAXOBJ; i++)
115         if (m == obj[i][0].use)
116             return i;
117
118     return -1;
119 }
120
121 // 区分け
122 void pigeonhole(OBJECT *tmp, int tmpcnt)
123 {
124     int i, objnum, tail_n, first, last, val;
125
126     // prev_use取得
127     for (objnum = 0; 0 <= objnum && objnum < MAXOBJ; objnum++)
128         prev_use[objnum] = obj[objnum][0].use;
129
130     // obj[A][0].useが1のAが存在すればそこから1cm未満か検索
131     // 次がuse0もしくは[0]ならそこをエンドポイントに
132     while (tmpcnt > 0) {
133         while (!tmp[tmpcnt].use)    tmpcnt--;
134         first = (tmpcnt < 0) ? 0 : tmpcnt;
135         while (tmp[tmpcnt].use)    tmpcnt--;
136         last = (tmpcnt < 0) ? 0 : tmpcnt+1;
137 #ifdef PRN
138         printf("first:%d last:%d\n", first, last);
139 #endif
140         val = first-last+1;
141
142         if (val == 1) {
143 #ifdef PRN
144             printf("NOISE CLEAR\n");
145 #endif
146             continue;    // ノイズ除去
147         }
148
149         // ここからtmpがどのobj[A]に近いかを判別する
150         for (objnum = 0; 0 <= objnum && objnum < MAXOBJ; objnum++) {
151 #ifdef PRN
152             printf("NEAR SEARCHING: objnum:%d\n", objnum);
153 #endif
154             tail_n = obj[objnum][0].use;
155
156             // 空のオブジェクトは検査しない
157             if (tail_n == 0) {
158 #ifdef PRN
159                 printf("EMPTY OBJECT\n");
160 #endif
161                 continue;
162             }
163
164             for (i = first; i != last; i--) {

```

```

165         if (!within_range (tmp[i], obj[objnum][tail_n], 0.05)) {
166 #ifdef PRN
167             printf("NO MATCH(OUTRANGE):\n"
168                 "tmp(%f,%f)\tobj(%f,%f)\n"
169                 , tmp[i].x, tmp[i].y, obj[objnum][tail_n].x, obj[objnum][tail_n].y);
170 #endif
171             continue;    // no match
172         }
173
174         // OBJBUF-tail_n-1とfirst-last+1を比較して
175         // memcpy(memory.hが必要?)
176         if (OBJBUF-tail_n-1 > first-last+1) {    // 空き有
177 #ifdef PRN
178             printf("EMPTY\n");
179 #endif
180             memcpy(&obj[objnum][tail_n+1], &tmp[last], sizeof(OBJECT)*val);
181             obj[objnum][0].use += val;
182 #ifdef OBJCAP
183             printf("-----\nCAPTURE\n");
184             printf("tail_n: %d\n", tail_n);
185             for (i = 1; i <= obj[objnum][0].use; i++)
186                 printf("%d:\t%f\t%f\n" , i, obj[objnum][i].x, obj[objnum][i].y);
187             printf("-----\n");
188 #endif
189         } else {
190 #ifdef PRN
191             printf("FULL\n");    // 空き無
192 #endif
193             memcpy(&obj[objnum][1], &tmp[last], sizeof(OBJECT)*val);
194             obj[objnum][0].use = val;
195         }
196         objnum = -2;    // break後メイン処理ループも抜けるため
197         break;
198     }
199 }
200
201 // tmpが既に登録されているobj[A]にヒットした
202 if (objnum != MAXOBJ)    continue;
203
204 // tmpがどのobj[A]にも近くなかった
205 #ifdef PRN
206     printf("NO NEAR OBJECT val: %d\n", val);
207 #endif
208     if (val < 5) {
209 #ifdef PRN
210         printf("NOISE CLEAR\n");
211 #endif
212         continue;    // ノイズ除去
213     }
214
215     // 必要かもしれない
216     //for (objnum = 0; objnum < MAXOBJ; objnum++)
217     for (objnum = 0; 0 <= objnum && objnum < MAXOBJ; objnum++) {
218 #ifdef PRN
219         printf("objnum: %d : use %d\n", objnum, obj[objnum][0].use);
220 #endif

```

```

221         tail_n = obj[objnum][0].use;
222         if (tail_n == 0) {
223 #ifdef PRN
224             printf("NEWOBJ CREATED objnum: %d\n", objnum);
225 #endif
226             memcpy(&obj[objnum][1], &tmp[last], sizeof(OBJECT)*val);
227             obj[objnum][0].use = val;
228             objnum = -1;
229             break;
230         }
231     }
232
233     if (objnum == MAXOBJ) { // 空きがなかった
234         for (i = 1; i < MAXOBJ; i++)
235             discard_obj(i);
236
237         break;
238     }
239 }
240
241 for (objnum = 0; objnum < MAXOBJ; objnum++)
242     if (obj[objnum][0].use == prev_use[objnum])
243         discard_obj(objnum);
244
245 // tmpは初期化しなくても上書きされる
246
247 return;
248 }
249
250 // PIを法としたthetaの絶対値が0.01未満であれば半周
251 int half_circle(double theta)
252 {
253     return (fmod(fabs(theta), M_PI) < 0.01) ? 1 : 0;
254 }
255
256 int main(int argc, char *argv[])
257 {
258     S2Port *port;    // デバイスポート
259     S2Sdd_t buf;     // データ取得用ダブルバッファ
260     S2Scan_t *scan;  // データ読み出し用構造体
261     S2Param_t param; // センサのパラメータ構造体
262
263     int i, j, ret, human = -1;
264     double pos_x_gl, pos_y_gl, pos_theta_gl; // (m), (rad) GL座標系のロボットの位置
265
266     if (argc != 2) {
267         fprintf(stderr, "USAGE: %s device\n", argv[0]);
268         return 0;
269     }
270
271     // Spur初期化
272     if (Spur_init() < 0) {
273         fprintf(stderr, "ERROR : cannot open spur.\n");
274         return -1;
275     }
276

```

```

277     Spur_set_pos_GL(0, 0, 0);
278     Spur_set_pos_LC(0, 0, 0);    // 最初だけ
279     Spur_set_vel(0.1);
280     Spur_set_accel(0.5);
281     Spur_set_angvel(M_PI/2);
282     Spur_set_angaccel(M_PI/8);
283
284     // Scip初期化
285     port = Scip2_Open(argv[1], B0);
286     if (port == 0) {
287         fprintf(stderr, "ERROR: Failed to open device.\n");
288         return 0;
289     }
290     printf("Port opened\n");
291     signal(SIGINT, ctrlc);
292     S2Sdd_Init(&buf);
293     printf("Buffer initialized\n");
294
295     // 初期化
296     escape = 0;
297
298     // オブジェクト初期化
299     for (i = 0; i < MAXOBJ; i++)
300         for (j = 0; j < OBJBUF; j++)
301             obj[i][j].use = obj[i][j].x = obj[i][j].y = 0;
302
303     // URGのパラメータ取得
304     Scip2CMD_PP(port, &param);
305
306     // URG-04LXの全方向のデータを取得開始
307     Scip2CMD_StartMS(port, param.step_min, param.step_max,
308         1, 0, 0, &buf, SCIP2_ENC_3BYTE);
309
310     while (!escape) {
311         ret = S2Sdd_Begin(&buf, &scan);
312         if (ret > 0) {
313             OBJECT tmp[scan->size];
314             double pole_x, pole_y, pole_r;
315             int obj_m = 0;
316
317             double d, theta;    // (mm), (rad) URGの生データ(極座標系)
318             double x_urg, y_urg;    // (m) URGのデータ(センサ座標系)
319             double x_fs, y_fs;    // (m) FS座標系のURGのデータ(ロボット座標系)
320             double x_gl, y_gl;    // (m) GL座標系のURGのデータ(世界座標系)
321             double x_lc, y_lc;    // (m) LC座標系のURGのデータ(ポール座標系)
322
323             // ロボットの現在位置を取得
324             Spur_get_pos_GL(&pos_x_gl, &pos_y_gl, &pos_theta_gl);
325
326             // センサデータをGL座標系に張り付け
327             for (i = 0; i < scan->size; i++)
328             {
329                 // 極座標系のデータの取得
330                 d = scan->data[i];
331                 theta = (double)(param.step_min + i - param.step_front)
332                     * 2.0 * M_PI / param.step_resolution;

```

```

333
334      ///// ものに当たりそうになったらバックする機能
335      //printf("DDDD %f\n", d);
336      //if (scan->size/2-5 < i && i < scan->size/2+5 && 0 < d && d < 10) {
337      //    Spur_stop();
338      //    Spur_vel(-0.2, pos_theta_gl);
339      //    usleep(100000);
340      //}
341
342      // printf("DEBUG用 theta: %f\n", theta);
343      // Pole_setupではthetaの一部だけ使うようにする
344      // thetaがある範囲にいるときcontinue
345      if ((state == Pole_setup && theta < 0) // 左しか見ない
346          || d < param.dist_min || d > 2000) { // maxだと5.6mも！！
347          tmp[i].use = 0;
348  #ifdef RAWTEST
349          printf("----\t----\n");
350  #endif
351          continue;
352      }
353
354      // 極座標系からセンサ座標系への変換
355      x_urg = d * cos(theta) * 0.001;
356      y_urg = d * sin(theta) * 0.001;
357
358      // センサ座標系からFS座標系への変換
359      x_fs = x_urg;
360      y_fs = y_urg;
361
362      // FS座標系からGL座標系への変換
363      x_gl = pos_x_gl + x_fs * cos(pos_theta_gl)
364            - y_fs * sin(pos_theta_gl);
365      y_gl = pos_y_gl + y_fs * cos(pos_theta_gl)
366            + x_fs * sin(pos_theta_gl);
367
368      // GL座標系からLC座標系への変換
369      gl_to_lc(x_gl, y_gl, &x_lc, &y_lc);
370      tmp[i].x = x_lc;
371      tmp[i].y = y_lc;
372      tmp[i].use = 1;
373  #ifdef RAWTEST
374      printf("%f\t%f\n", x_lc, y_lc);
375  #endif
376      }
377  #ifdef TMPTEST
378      printf("TMP START\n");
379      for (i = 0; i < scan->size; i++) {
380          if (tmp[i].use)
381              printf("0A\t%f\t%f\n", tmp[i].x, tmp[i].y);
382          else
383              printf("----\t----\n");
384      }
385      printf("TMP END\n");
386  #endif
387      // センサが反応したGL座標をつなげ、配列として実装されたオブジェクトにする
388      // obj[0][]はボールの集合(全て隣り合っている)

```



```

389 // obj[1][]以降を隣り合った点の集合とする
390
391 // 次のサイクルでその周辺になければ消す(useを0に)
392 // obj[0][]とかは[0][0]に100.0以外の何かが入っているとき
393 // 1回で複数検出されないものは明らかにノイズなので、tmp[]は毎回リセットする
394 // tmp[]の類似のやつを探してあれば全部追加する、なければ削除する
395 // 消されるのはobj[0][],[1][]だけ。
396 // obj[1][]はAlien->Patrolになるときリセットする。
397
398 // 仮定: 同じ物体はtmp配列の連続したところに保存される
399
400 pigeonhole(tmp, scan->size);
401
402 double lcx, lcy, lct;
403 if (state == Pole_setup) {
404     Spur_free();
405
406     obj_m = mostuse();
407     if (obj[obj_m][0].use < 500) { // 500までたまっていない
408         S2Sdd_End(&buf);
409         continue;
410     }
411
412 #ifdef PRN
413     printf("-----\nMAYBE A POLE obj_m: %d\n", obj_m);
414     for (i = 1; i < OBJBUF; i++)
415         if (obj[obj_m][i].use)
416             printf("%3d:\t%f\t%f\n" , i, obj[obj_m][i].x, obj[obj_m][i].y);
417     printf("-----\n");
418     for (i = 1; i < MAXOBJ; i++)
419         printf("obj[%d]:\t%d\n", i, obj[i][obj_m].use);
420     printf("-----\n");
421 #endif
422
423 // 最も反応しているobjの円の方程式を出す
424 p3_to_circle(obj[obj_m], &pole_x, &pole_y, &pole_r);
425
426 if (pole_r < POLE_R) {
427 #ifdef PRN
428     printf("-----\n\nGET A POLE!!!\n");
429     printf("%f\t%f\t%f\n" , pole_x, pole_y, pole_r);
430     printf("-----\n");
431 #endif
432
433 //最終的にはLCをセットして完了
434 Spur_set_pos_LC(-(pole_x-pos_x_gl), -(pole_y-pos_y_gl)
435                 , pos_theta_gl);
436 state = Patrol;
437
438 // 全部消す
439 for (i = 0; i < MAXOBJ; i++)
440     discard_obj(i);
441 } else if (state == Patrol) {
442 #ifdef STATE
443     printf("STATE IS --- PATROL ---\n");
444 #endif

```

```

445 // ***周回***
446 Spur_circle_LC(0, 0, DISTANCE);
447
448 // human 判定
449 for (i = 0; i < MAXOBJ; i++) {
450     // 100以上たまっていなければ無視
451     if (obj[i][0].use < 100) continue;
452
453     //gl_to_lc(obj[i][300].x, obj[i][300].y, &lcx, &lcy);
454     //double alien_dist = pow(lcx, 2.0) + pow(lcy, 2.0);
455
456     // 原点からroot2m以上離れている
457 #ifdef PRN
458     printf("GIWAKU%f\t%f\t%f\n", obj[i][100].x, obj[i][100].y, pow(obj[i][100].x, 2.0)
459 #endif
460     if (1 <= pow(obj[i][100].x, 2.0) + pow(obj[i][100].y, 2.0)) {
461 #ifdef PRN
462         //printf("alien (%f, %f)\n", lcx, lcy);
463         //printf("CATCH MODE i:%d\n", i);
464 #endif
465         human = i;
466         human_x = obj[human][100].x;
467         human_y = obj[human][100].y;
468         human_t = atan2(human_y, human_x);
469
470         state = Alien_found;
471         for (i = 0; i < MAXOBJ; i++)
472             discard_obj(i);
473         break;
474     }
475 }
476
477 // 周回軌道は1周ごとに修正する
478 // obj[0][]を利用する
479 // 3点取ってきて中心を計算、半径は初期情報として与えられる。
480 // (境界判定を簡単化するため3/4周ごとでも良い)
481 // 半周ごとにthetaがマイナスになることを利用する
482 // 半周ごとにobjはリセットした方が綺麗に出るかも?
483 // (曲がる時のズレが大きいようなので)
484 // NULLで問題ないようなら47行目も修正
485 Spur_get_pos_LC(&lcx, &lcy, &lct);
486
487 // 半周したならば
488 if (half_circle(lct)) {
489 #ifdef PRN
490     printf("-----\n\n\n\n\n\nADJUST");
491     printf("\n\n\n\n\n\n-----\n");
492 #endif
493     obj_m = mostuse();
494
495     int u = obj[obj_m][0].use;
496     double d = pow(obj[obj_m][u].x, 2.0) + pow(obj[obj_m][u].y, 2.0);
497
498     // 原点からの距離的にmostuseがボールではなければ抜ける
499     if (0.5 <= d) break;
500

```

```

501         p3_to_circle(obj[obj_m], &pole_x, &pole_y, &pole_r);
502         Spur_set_pos_LC(-pole_x, -pole_y, lct);
503
504         // 全部消す
505         for (i = 0; i < MAXOBJ; i++)
506             discard_obj(i);
507     }
508     } else if (state == Alien_found) {
509 #ifdef STATE
510         printf("STATE IS --- FOUND ---\n");
511 #endif
512         Spur_stop_line_LC(human_x/2, human_y/2, human_t);
513         usleep(10000);
514
515         //単位の確認
516         if (Spur_near_pos_LC(human_x/2, human_y/2, 1)) // 単位の確認
517             state = Alien_chase;
518
519         // 不審者のいる場所（中心点）を見つけたら
520         // そととボール(直径12cm)の間に十分な距離があるかを判定し、
521         // あれば割り込んでいく。なければ近づいて警告
522         //obj_m = mostuse();
523         //OBJECT o = obj[obj_m][obj[obj_m][0].use];
524         //if (pow(o.x, 2.0) + pow(o.y, 2.0) > 0.5 && obj[obj_m][0].use) {
525         //    human = -1;
526         //    state = Alien_lost;
527         //}
528         /*
529         } else if (Spur_near_pos_LC(0, 0, LOSTAREA) == 1) {
530             // ロボットがLOSTAREAを超えたらロスト、ボールの周回に戻る
531 #ifdef PRN
532             printf("HUMAN!!! human:%d\n", human);
533 #endif
534             Spur_line_LC(human_x, human_y, human_t);
535
536             //printf("\n-----\n");
537             //for (i = 1; i < OBJBUF; i++)
538             //    printf("obj[human][%d]:\t(%f,\t%f)\n", i, obj[human][i].x, obj[human][i].y);
539             //printf("\n-----\n");
540
541
542             //Spur_stop_line_GL(human_x/2, human_y/2, 0);
543         } else {
544             human = 0;
545             state = Alien_lost;
546         }
547     */
548     } else if (state == Alien_chase) {
549 #ifdef STATE
550         printf("STATE IS --- CHASE ---\n");
551 #endif
552         // 追いかけるならこの部分を記述する
553         obj_m = mostuse();
554         OBJECT o = obj[obj_m][obj[obj_m][0].use];
555
556         Spur_stop_line_LC(o.x*3/5, o.y*3/5, atan2(o.y, o.x));

```

557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612

```
        usleep(10000);

// 見失ったor追い出した
if (Spur_near_pos_LC(0, 0, LOSTAREA)) {
    human = -1;

    // 全部消す
    for (i = 0; i < MAXOBJ; i++)
        discard_obj(i);
    state = Alien_lost;
}
for (i = 0; i < MAXOBJ; i++) {
    // 100以上たまっていなければ無視
    if (obj[i][0].use < 100) continue;

    // 原点からroot2m以上離れている
    printf("KIETAKA%f\t%f\t%f\n", obj[i][100].x, obj[i][100].y, pow(obj[i][100].x, 2.0) + pow(obj[i][100].y, 2.0));
    // まだ不審者がいる
    if (1 < pow(obj[i][100].x, 2.0) + pow(obj[i][100].y, 2.0)) {
        break;
    } else if (i == MAXOBJ-1) {
        human = -1;

        // 全部消す
        for (i = 0; i < MAXOBJ; i++)
            discard_obj(i);
        state = Alien_lost;
    }
}

//if ((pow(o.x, 2.0) + pow(o.y, 2.0) > 0.5 && obj[obj_m][0].use < 1)
// || Spur_near_pos_LC(0, 0, LOSTAREA)) {
//    human = -1;

// // 全部消す
// for (i = 0; i < MAXOBJ; i++)
//     discard_obj(i);
// state = Alien_lost;
//}
} else if (state == Alien_lost) {
#ifdef STATE
    printf("STATE IS --- LOST ---\n");
#endif

    // 普通に周回軌道に戻るだけでうまくいきそう
    // つまりcircle_LCするだけにしてしまう
    //
    // -> うまくいかなかった
    //
    Spur_circle_LC(0, 0, DISTANCE);
    usleep(10000);
    state = Patrol;

    // 近くに着いた
    //
    //if (Spur_near_pos_LC(0, 0, 0.5)) {
```

```

613         // state = Patrol;
614     //} else {
615     // // 全部消す
616     // for (i = 0; i < MAXOBJ; i++)
617     //     discard_obj(i);
618     //}
619
620     // obj_m = mostuse();
621     // if (!obj[obj_m][500].use) {
622     //     S2Sdd_End(&buf);
623     //     continue;
624     // }
625
626     // // [0].useが多いやつを送るように変更する
627     // p3_to_circle(obj[obj_m], &pole_x, &pole_y, &pole_r);
628
629     // // ボールが見つかったら
630     // if (pole_r < POLE_R) {
631     //     全部消す
632
633     //     最終的にはLCをセットして完了
634
635     //     Spur_set_pos_LC(-(pole_x-pos_x_gl), -(pole_y-pos_y_gl), pos_theta_gl);
636     //
637     //     周回軌道に戻すだけならnear_pos判定に意味はない
638     //     if (Spur_near_pos_LC(0, 0, 10) == 1)
639     //         state = Patrol;
640     // }
641 }
642
643 // S2Sdd_BeginとS2Sdd_Endの間でのみ、構造体scanの中身にアクセス可能
644 S2Sdd_End(&buf);
645 }
646 else if(ret == -1) {
647     // 致命的なエラー時(URGのケーブルが外れたときなど)
648     fprintf(stderr, "ERROR: Fatal error occurred.\n");
649     break;
650 }
651 else {
652     // 新しいデータはまだ無い
653     usleep(100000);
654 }
655 }
656
657 Spur_set_vel(0.0);
658 Spur_set_accel(-1.0);
659 printf("\nStopping\n");
660
661 ret = Scip2CMD_StopMS(port, &buf);
662 if (ret == 0) {
663     fprintf(stderr, "ERROR: StopMS failed.\n");
664     return 0;
665 }
666
667 printf("Stopped\n");
668 S2Sdd_Dest(&buf);

```

```
669     printf("Buffer destructed\n");
670     Scip2_Close(port);
671     printf("Port closed\n");
672
673     return 1;
674 }
```

Listing 7.5 最終課題 5「警備員」