

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 . . . . .	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 . . . . .	9
6.2.3	パトロール . . . . .	9
6.2.4	不審者発見 . . . . .	9
6.2.5	不審者追従 . . . . .	9
6.2.6	不審者逃走 . . . . .	9
6.3	結果 . . . . .	9
6.4	考察 . . . . .	9

# 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 X)
カーネル	Linux 4.4.0-71
OS	Ubuntu13.10
YP-Spur	

## 1.2 学んだこと

## 1.3 謝辞

TA の方々には非常によくお世話になりました。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

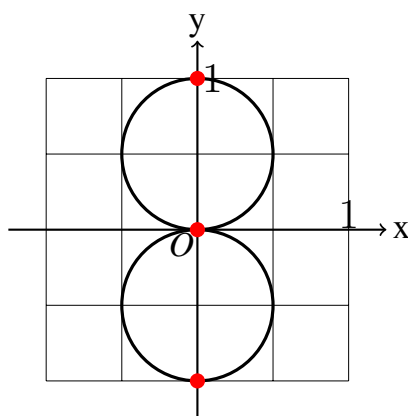


図 2.1 原点を出発して8の字を描くように進む

### 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。

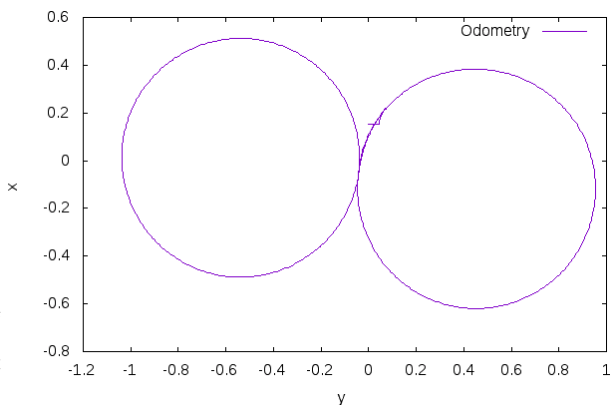


図 2.2 8の字走行のオドメトリ

## 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

### 3.2 解法

### 3.3 結果

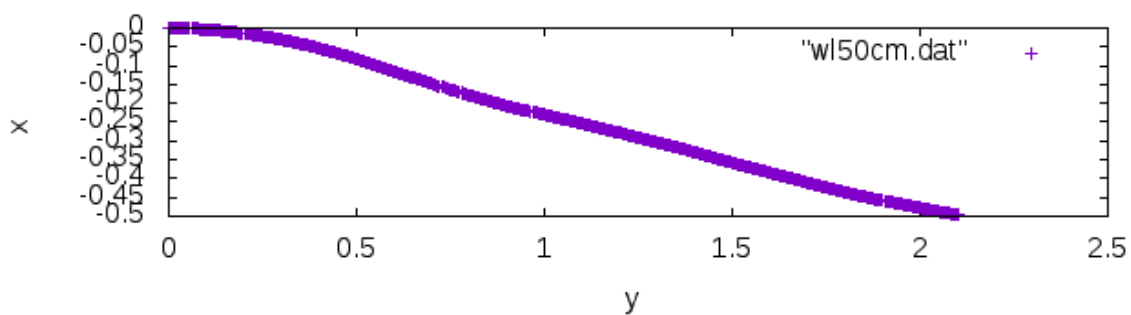


図 3.1 1m 先に壁を見つけて停止するまでのオドメトリ

小刻みに調整が働いて壁と並行する線上を走行できていることがわかる。

### 3.4 考察

## 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。

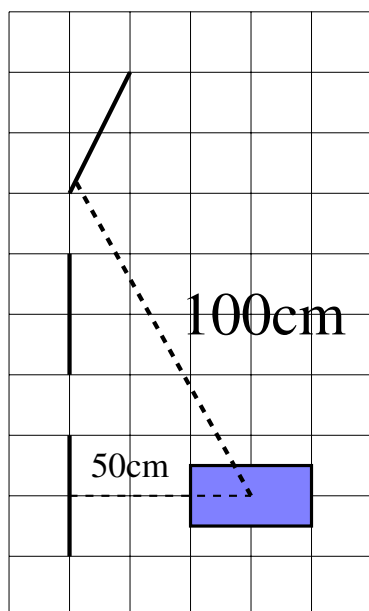


図 4.1 センサデータは左前方の一点を利用する

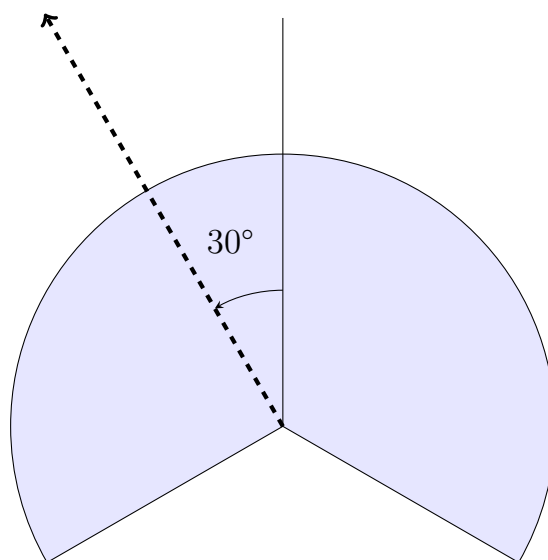


図 4.2 センサ範囲

もやし

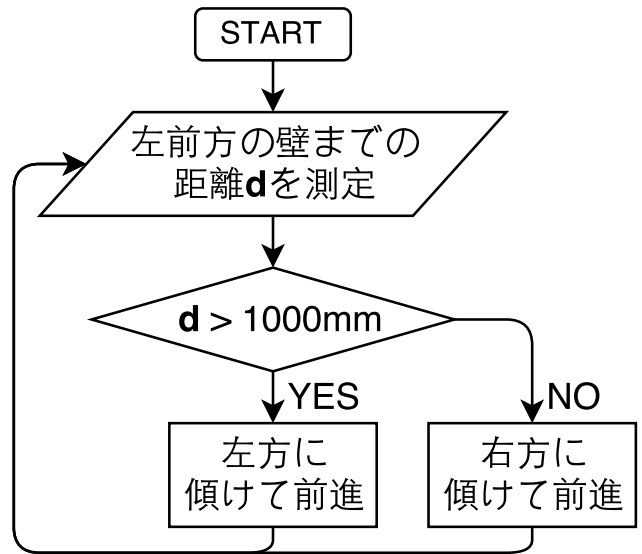


図 4.3 壁並走アルゴリズム案

#### 4.3 結果

#### 4.4 考察



## 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。図では X や Y の表示が逆転しているものもあるが、軸に関しては正しく表示しているのでパラメータの仮変数だと思って無視していただきたい。

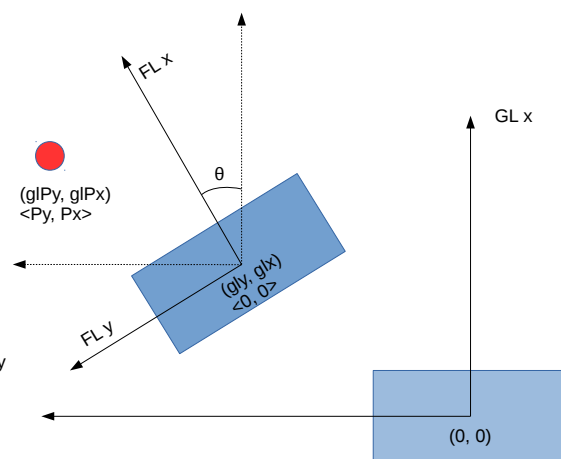


図 5.1 FS 座標系と GL 座標系の関係

<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。

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。

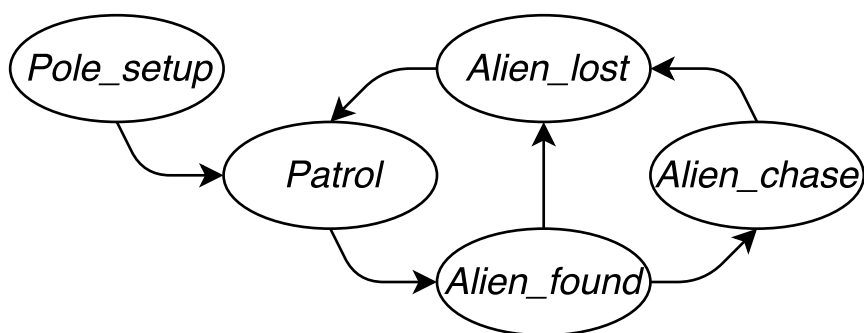


図 6.1 作成したプログラムの状態遷移図

#### 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

状態遷移図では *Pole\_setup* に対応する。

ポールの探索段階では人や壁などのもの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。

オブジェクトの中で

全状態で唯一 GL 座標を基準に動作する。

### 6.2.3 パトロール

状態遷移図では *Patrol* に対応する。

### 6.2.4 不審者発見

状態遷移図では *Alien\_found* に対応する。

### 6.2.5 不審者追従

状態遷移図では *Alien\_chase* に対応する。

### 6.2.6 不審者逃走

状態遷移図では *Alien\_lost* に対応する。

戻ってくる。

## 6.3 結果

## 6.4 考察