2003.2.11

2002 年度ＩＳテーマ

H8-3048F と C 言語を用いた自立型迷路探索ロボットの開発

情報工学科　r00h0390　藤川雅男

１．はじめに

１-１．本研究の目的

本研究の目的は、マイコン応用システムを利用した自立型迷路探索ロボット「マイクロマウス」の製作を通して、電気・電子・論理回路、マイクロコンピュータといったものへの理解を深め、電子工作およびアセンブラ言語プログラミング技能の向上を図ることである。

研究題材としてマイクロマウスを選んだ理由は、

○ コンピュータ・センサ・アクチュエータの諸部分の総合であり、メカトロニクス技術の基礎を体系的に学ぶことができる。
○ 全国に初心者からベテランまでの愛好家が数多くおり、入門しやすい。また、ハードウェア・ソフトウェアともに適度に難解で奥が深い。
○ 競技化されており、はっきりした目標にむけて努力することができる。

というものである。

今回は、2001 年の全国大会の反省を元にして新しく機体を製作し、名古屋で行なわれた「マイクロマウス中部地区大会」エキスパートクラス、東京で行なわれた「マイクロマウス 2002」エキスパートクラスの試合に出場・完走することを目標とした。

なお、前回は CPU として Z80 を、開発言語として Z80 アセンブラを用いたが、今回は新しい試みとして CPU に H8-3048F、開発言語として C 言語を用いることにした。また、前回迷路探索はできたが最短経路の探索にまでは手が回らなかったので、今回はそこに重点を置いてプログラム開発を行った。

１-2．マイクロマウス 2002 とは

マイクロマウスとは、マイコン応用システムを利用して迷路探索・走行をする自立型ロボットのことである。16×16 のます目の中央４ますをゴールとし、スピードを競う競技としてニューテクノロジー振興財団マイクロマウス委員会により全国大会が開かれている。2002 年には東京「科学技術館」にて、11 月 9・10 日に行なわれた。

２．研究方法

2-1. マシンデータ

研究場所	制御通信部部室（洛北校 F 教室裏）
作成者	藤川雅男
マシン名	肉弾激餓
命名の由来	劇画が好きだから。
車体	昨年度の機体を改造して作成。
ー筐体	アルミ、部分的にガラスエポキシ板を使用。一部マイクロマウス委員会東日本支部標準部品を利用した。
ー車輪	市販のアルミ製ホイール。上に両面テープで粘性の高いゴムシート（田宮 RD タイヤキャッププラバー（ソフト）OP-184）を貼り付けた。
CPU	H8-3048F（秋月電子 AKI-H8 3048F）を使用。
アクチュエータ	ステッピングモータ 2 つ（SANYO103-540-36）
センサ	壁上面検出タイプ（ウィング形)。反射型フォトインタラプタ 14 個（オムロン EE-SY121。前 2 つ、左右各 6 つ）使用
電源	ニッケル水素電池単 3 型 1.2V、1700mAh（GP 社 170AAHC）12 本（14.4 ボルト)、3 端子レギュレータ（2405A）で降圧。
プログラム開発の方法	C 言語による。コーディングはベストテクノロジー社の GCC Developer Lite を使用。 AKI-H8 へのファイルの転送には FlashWriter2.6 を使用。 転送ケーブルは RS232C ケーブルの一端をステレオジャックに付け替えたものを使用。受け側には、AKI-H8 の、脚の一部を上向きに取り付け、これに別に製作した転送ボードを挿すという形をとった。 探索アルゴリズムの研究には自作の迷路探索シミュレータを使用。

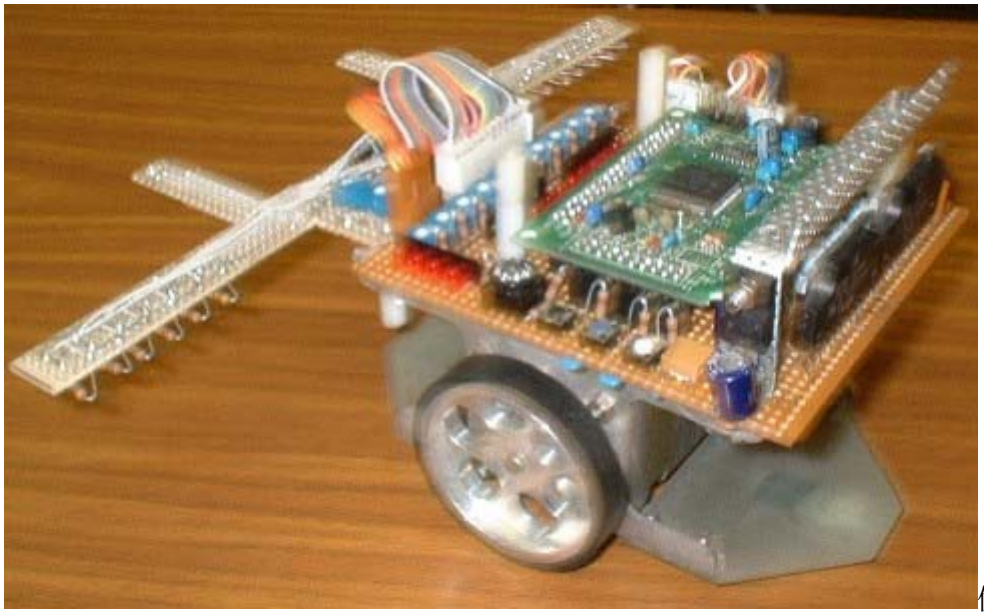
2-2. 開発の方針

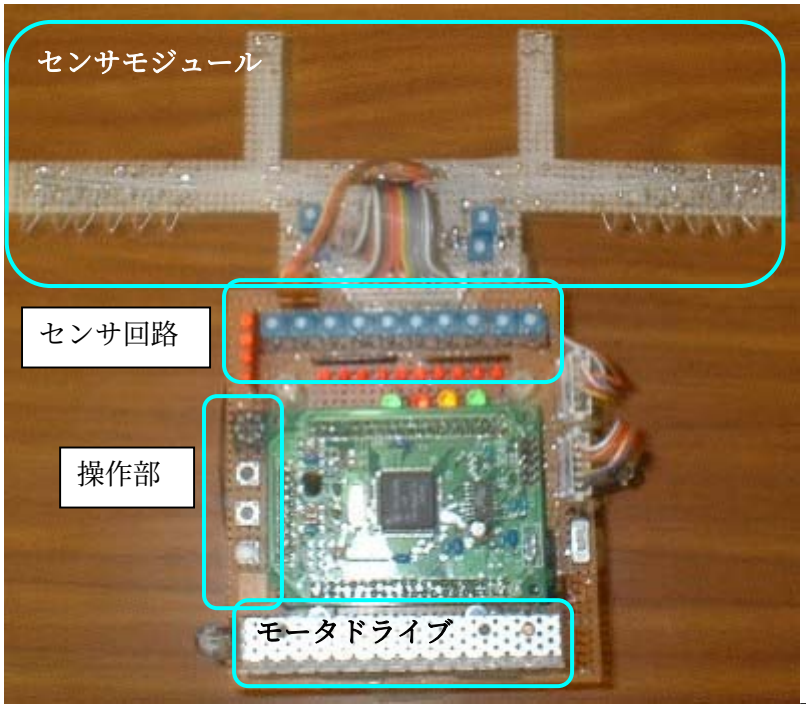
マイクロマウス 2001 での反省から、次のような作成方針を立てた。

迷路探索・最短経路計算ができなかったため、今回はソフトウェア開発に力を入れることにする。そのための方法として、Microsoft Visual C++6.0 を用い、Windows 上で動作するシミュレータを作製する。この上で十分に迷路探索・最短経路計算アルゴリズムのデバッグを重ねた上で、これを H8 に移植する。

３．内容詳細

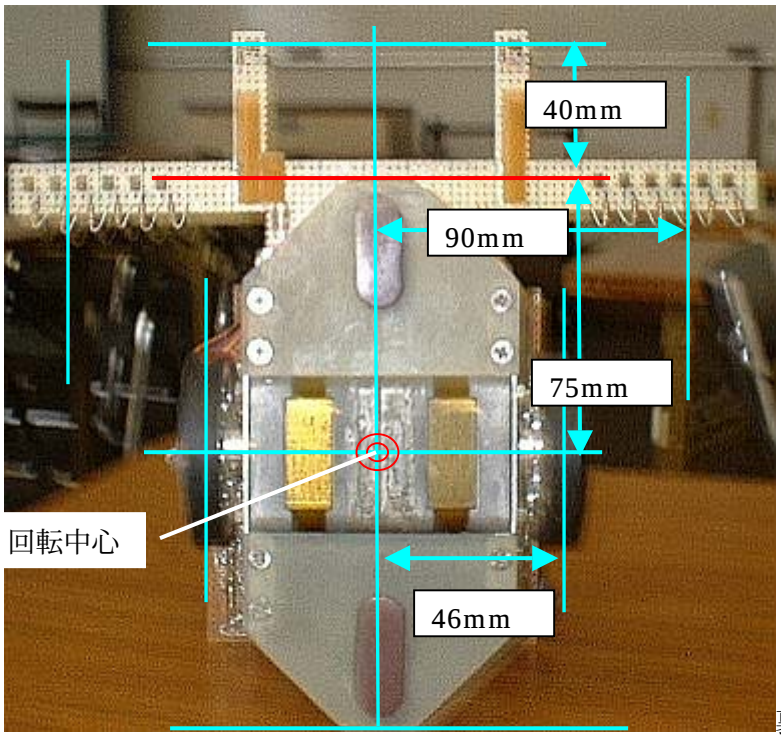
3-1. 車体


俯瞰

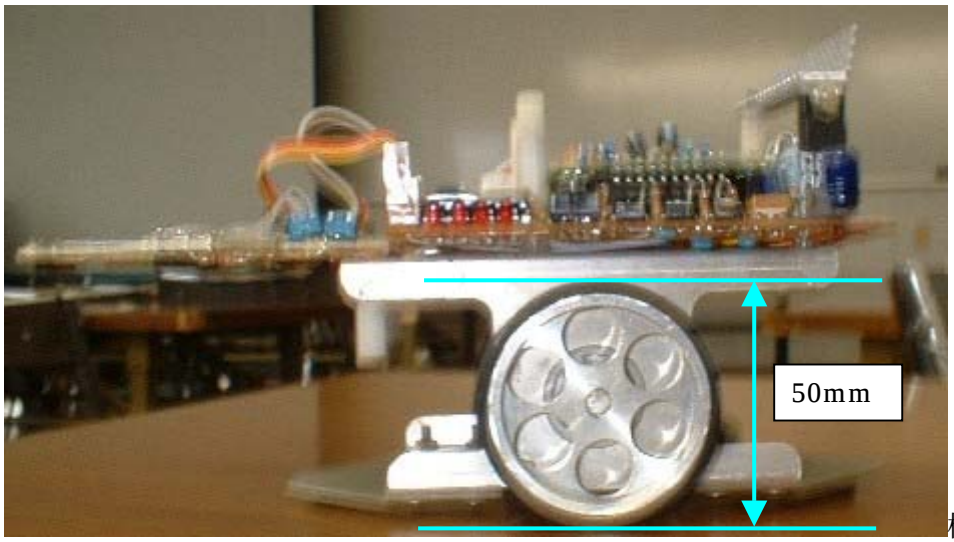


上面

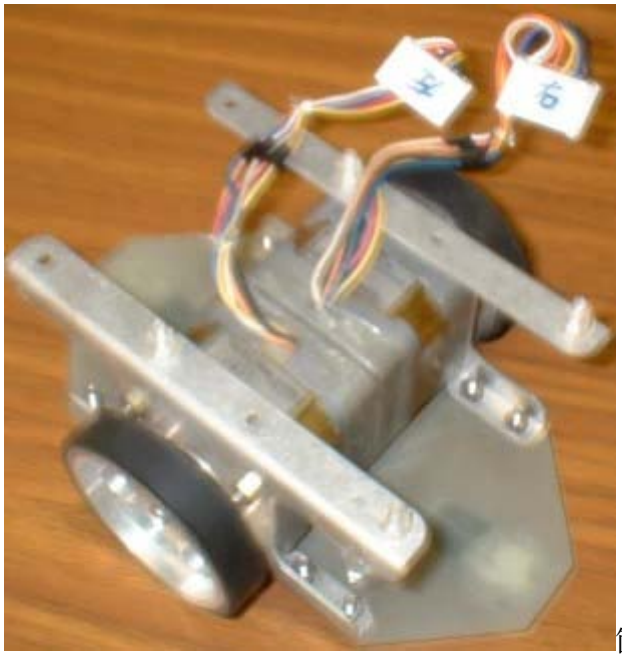
車体はセンサモジュールとその他の回路部、筐体の三つの部分に分けることができる。



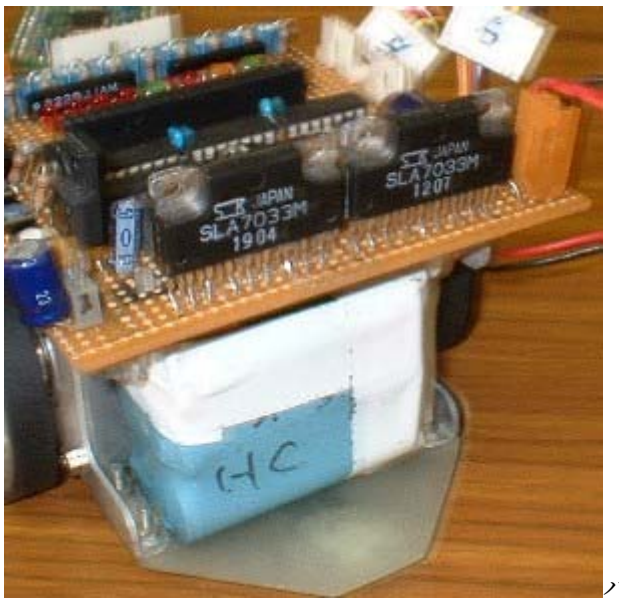
裏面



横面


筐体

筐体の「工」型の部品はマイクロマウス委員会東日本支部標準部品を利用している。底板にはガラスエポキシ板を使用。



バッテリーパックを積む場所

バッテリーは単三電池を 6 本ずつ組み合わせたものを二つ用いている。これらを直列につなぐことにより、モータ回転に使用する 14.4 ボルトの電圧を得ている。また、同じ電源を 3 端子レギュレータで 5 ボルトに降圧し、電子回路部分の電源としている。

3-2. CPU

CPU として、H8-3048F（AKI-H8 キット）を使用した。

3-2-1. H8-3048F の仕様

以下に、H8-3048F の仕様のうち今回関係が深い部分についてあげる。

CPU

- H8/300CPU に対してオブジェクトレベルで上位互換。
- 汎用レジスタは 16 ビット×16 本（8 ビット×16 本＋16 ビット×8 本、32 ビット×8 本としても使用可能）
- 最大動作周波数：16MHz
- 加減算：125[ns]
- 乗除算：875[ns]
- アドレス空間：16M バイト

メモリ

- ROM：128k バイト
- RAM：4k バイト

16 ビットインテグレーテッドタイマユニット（ITU）

- 16 ビットタイマ 5 チャンネルを内蔵、最大 12 端子のパルス出力、または最大 10 種類のパルスの入力処理が可能
- 16 ビットタイマカウンタ×１（チャネル 0～4）
- アウトプットコンペア出力、インプットキャプチャ入力（兼用端子）×２（チャネル 0～4）
- 同期動作可能（チャネル 0～4）
- PWM モード設定可能（チャネル 0～4）
- 位相係数モード可能（チャネル 3，4）
- バッファ動作可能（チャネル 3，4）
- リセット同期 PWM モード設定可能（チャネル 3，4）

- 相補 PWM モード設定可能（チャネル 3，4）
- コンペアマッチ、インプットキャプチャ A の割り込みにより DMAC 起動可能（チャネル 0～4）

シリアルコミュニケーションインタフェース（SCI）×2 チャネル

- 調歩同期、クロック同期式モードの選択可能
- 送受信同時動作
- 専用のボーレートジェネレータ内蔵
- スマートカードインタフェース拡張機能内蔵（SCI0 のみ）

I/O ポート

- 入出力端子 70 本
- 入力端子 8 本

動作モード

- アドレス空間やバス幅の初期値によって、7 種類の MCU 動作モードがある。

3-2-2. メモリ・I/O の増設について

今回メモリや I/O の増設は行わず、動作モードは 7 を用いた。

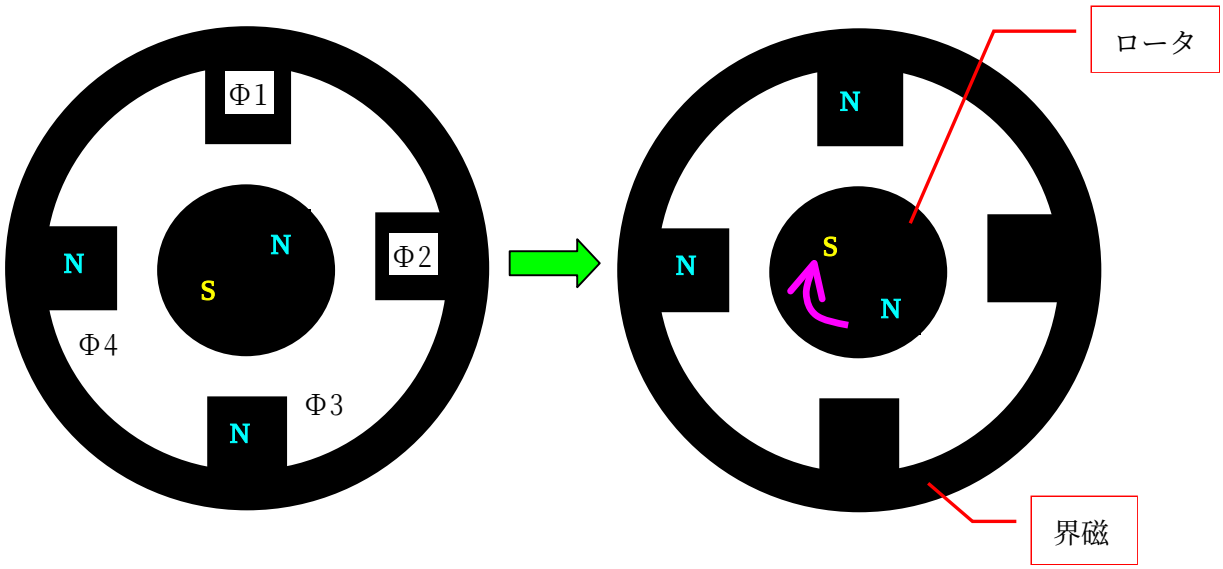
3-2-3. I/O の割り振り

P10：左センサ（左から一番目）入力	P70：左センサ（左から五番目）入力
P11：左センサ（左から二番目）入力	P71：左センサ（左から六番目）入力
P12：左センサ（左から三番目）入力	P72：右センサ（左から一番目）入力
P13：左センサ（左から四番目）入力	P73：右センサ（左から二番目）入力
P14：右センサ（左から三番目）入力	PA0：右モータ A 出力
P15：右センサ（左から四番目）入力	PA1：右モータ B 出力
P16：右センサ（左から五番目）入力	PA2：右モータ¥A 出力
P17：右センサ（左から六番目）入力	PA3：右モータ¥B 出力
P20：ロータリスイッチ１入力	PA4：左モータ A 出力
P21：ロータリスイッチ２入力	PA5：左モータ B 出力
P22：ロータリスイッチ４入力	PA6：左モータ¥A 出力
P23：ロータリスイッチ８入力	PA7：左モータ¥B 出力
P24：プッシュスイッチ１入力	PB0：モニタ LED１出力
P25：プッシュスイッチ２入力	PB1：モニタ LED２出力
P26：プッシュスイッチ３入力	PB2：モニタ LED３出力
P50：前センサ入力	PB3：モニタ LED４出力
P51：前センサ入力	PB4：モニタブザー出力

3-3. アクチュエータ

アクチュエータとして、ステッピングモータ（SANYO103-540-36）を二つ用いた。モータが古く弱っており、十分なパワーを出すことができなかったため、途中で同等品と取り替えたところうまく動いた。

3-3-1. ステッピングモータ回転の原理



(図は 2 相励磁方式)

ステッピングモータは、界磁が回転するように次々に励磁することによって、永久磁石でできたロータを回転させるものである。DC モータは電気を流せば回転するが、細かな制御を行うためにはロータリエンコーダで速度を測るなど、複雑な手順が必要となる。しかし、ステッピングモータは 1 パルスごとに進む距離が決まっているため位置の計算がたやすい。また、パルスの周波数に比例したスピードがでるため、加減速のプログラミングが容易である。PM 型、VR 型、HB 型など、いくつかの形式がある。今回使用したものは HB 型である。

励磁の方法には 1 相励磁方式、2 相励磁方式、1-2 相励磁方式がある。2 相励磁方式は、常に二つのコイルを励磁させる方法であり、電力を食うかわりに 1 相励磁方式に比べて安定した回転をさせることができる。1-2 相励磁方式は 1 相と 2 相を組み合わせた方式で、パルスごとの回転が半分になり、より細かい制御を行うことができる。今回は簡単な 2 相励磁方式を使った。

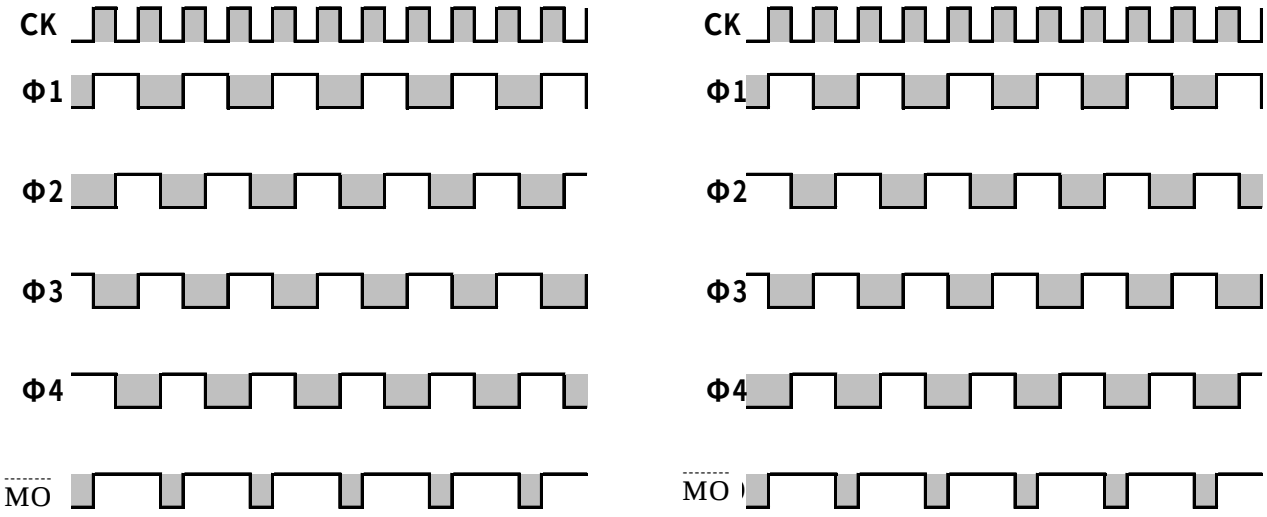
3-3-2. HB 型ステッピングモータの特徴

HB とは、Hybrid type の意である。界磁と対向したロータの外周に多数の歯車状突極があり、そこに軸方向に磁化された永久磁石が組み込まれている。界磁側も歯車状の形をなしており、二つのずれを利用して 1 パルスにつき 1.8 度ずつ（1-2 相励磁の場合は 0.9 度）

回転をさせることができる。

3-3-3. ステッピングモータに与えるパルス

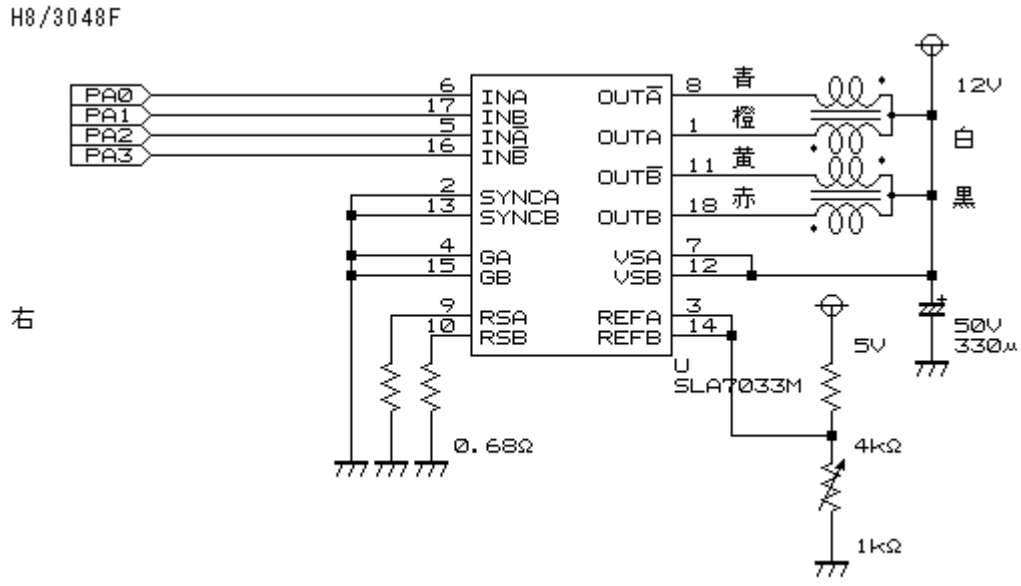
4つのコイルに対して与えるパルスのタイミングチャートは以下のとおりである。



(1)2相励磁正転　　(2)2相励磁逆転

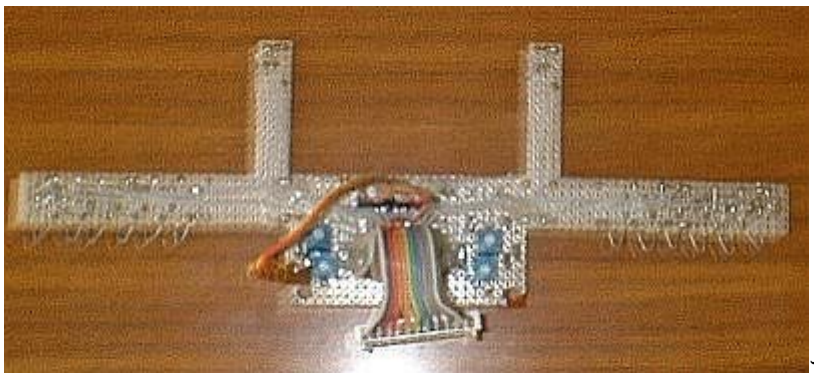
■は、ONの状態を表す。

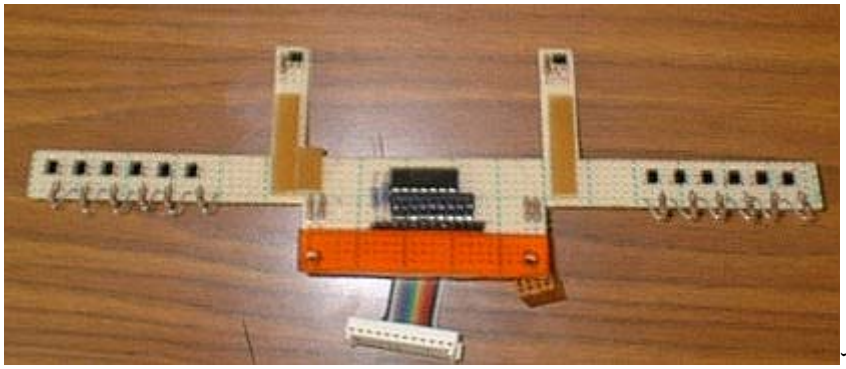
これらのパルス列はソフトウェア的にもハードウェア的にも発生させることができる。ハードウェアを用いる場合は、8713 等の専用 IC を用いると効率がよい。今回は専用 IC を用いず、ソフトウェアを用いて CPU から直接４つのパルス列を発生させた。これを増幅するにはトランジスタを用いる方法と専用の素子を用いる方法がある。今回は専用の素子 SLA7033M を用いて増幅し、モータに送った。



3-4. センサ

マイクロマウスは迷路をぶつからずに走るため、前と左右の壁を検知する必要がある。壁検知の方法としては、一般に反射式フォトインタラプタが用いられる。今回は小さく安価で感度の良いオムロンの EE-SY121 を用いた。

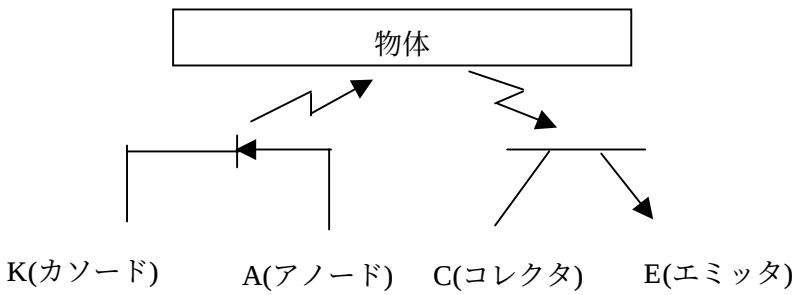

センサモジュール上面


センサモジュール裏面

並んでいる黒い粒状のものが EE-SY121 である。

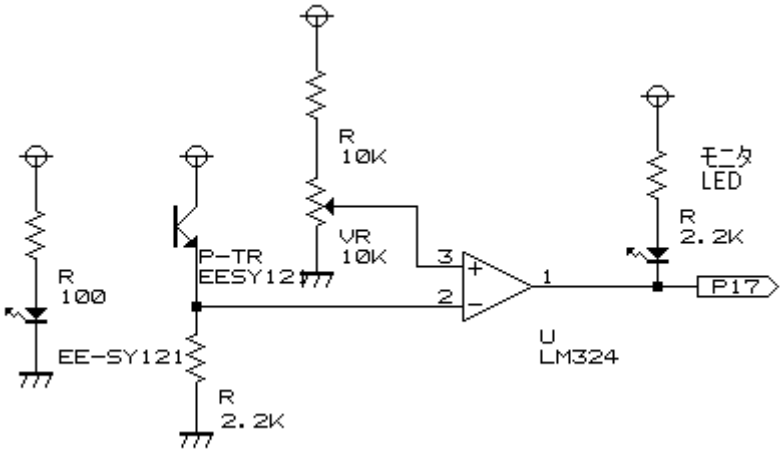
3-4-1. 反射式フォトインタラプタが壁を検出する原理

反射式フォトインタラプタは発光ダイオードとフォトトランジスタが一つの素子に組み合わされたものである。発光ダイオードが赤外線を発射し、反射光をフォトトランジスタが検出することにより、物体の存在を検知する。



今回、発光ダイオードは常時点灯とし、反射光を検知したフォトトランジスタの反応をオペアンプ（LM324）で増幅し、壁検出の方法としている。

フォトインタラプタ一つ分の回路は以下のとおりである。センサの感度は半固定抵抗 VR で調整する。



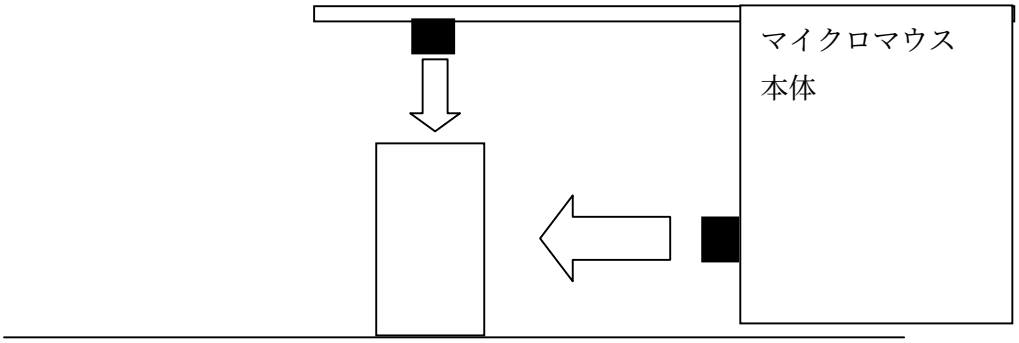
3-4-2. 壁の認識方法

壁の認識方法としては、

① 壁の上面を検出する方法（デジタルセンサ）

② 壁の側面を検出する方法（アナログセンサ）

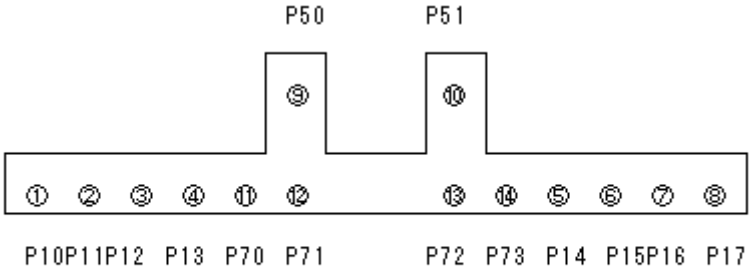
の 2 通りがある。



マシンを小さくまとめられる②の方式が近年の流行だが、難しいため、単純に壁の「ある・ない」を判断するだけで済む①の方式をとった。

3-4-3. フォトインタラプタの配置方法

今回は前に 2 つ、左右に 6 つずつ配置。上から見たとき、次のようになっている。



センサの配置

前センサが二つあるのは取りこぼしを防ぐためであり、左右のセンサが多くなっているのは、軌道修正のためである。当初横センサは①～⑧だけを使用していたが、軌道修正に難があるため⑪～⑭の４つを大会直前に追加した。

3-4-4. センサのテスト方法

まず、発光ダイオードが点灯しているかを確認する。赤外線は目に見えないため、ほんとうに点灯しているかどうか見ただけではわからない。確かめるには、デジカメを通してみればよい。赤外線が出ていれば白い光が出ているように見えるはずである。検出側が正しいかどうかを確認するには、モニタ LED を用いる。単体での動作を確認してから、CPU と接続すべきである。

3-5. 転送ケーブル

作成したプログラムをコンパイルした後、出来上がった mot 形式のファイルをマイコン側に転送しなければならない。その際、RS-232C ケーブルの一方の口をステレオジャック（オス）に換装したものを用いる。

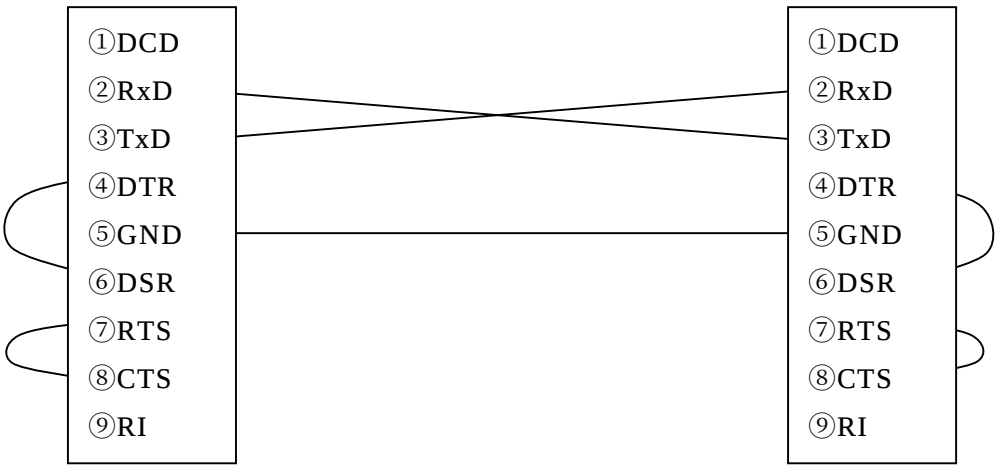
3-5-1. RS-232C とは

Recommended Standard 232 version C の略で、米国電子工業会(EIA)によって標準化された、シリアル通信の規格の一つである。シリアル通信方式としては最も普及しており、ほとんどのパソコンに標準で搭載されている。DOS/V 機のコネクタには D-sub9 ピンのものがよく使われている。パソコンとモデムなどとの接続に使われる。

D-sub9 の各ピンに流れる信号の名称と働きは以下のとおりである。

ピン番号	信号名	備考
1	DCD	キャリア検出
2	RxD	受信データ
3	TxD	送信データ
4	DTR	データ端末レディ
5	GND	シグナルグランド
6	DSR	データセットレディ
7	RTS	送信要求
8	CTS	送信可
9	RI	被呼表示

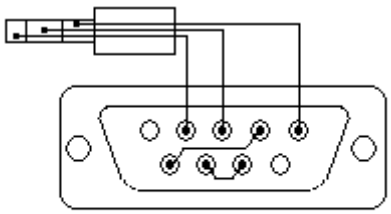
必ずしも、すべてのピンを使用する必要はない。最も簡単な接続方式は下記のようなものである。



3-5-2. ステレオピンジャックと D-SUB コネクタの接続方法

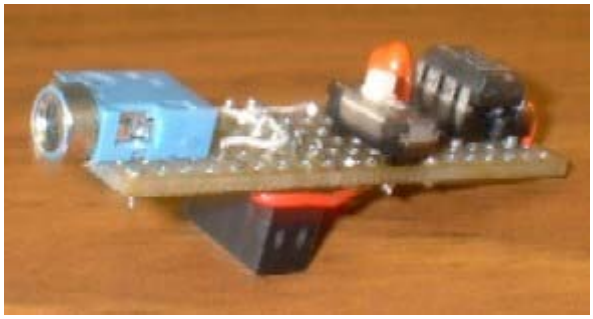
簡単な通信を行うためならば、RXD、TXD、GND の三つのピンだけでこと足りる。そのため、ロボットに搭載する口として簡便なステレオピンジャックを利用することが多い。

具体的な接続は下図のように行う。

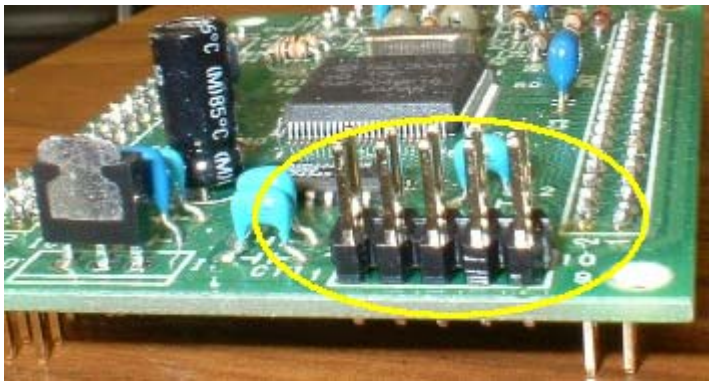


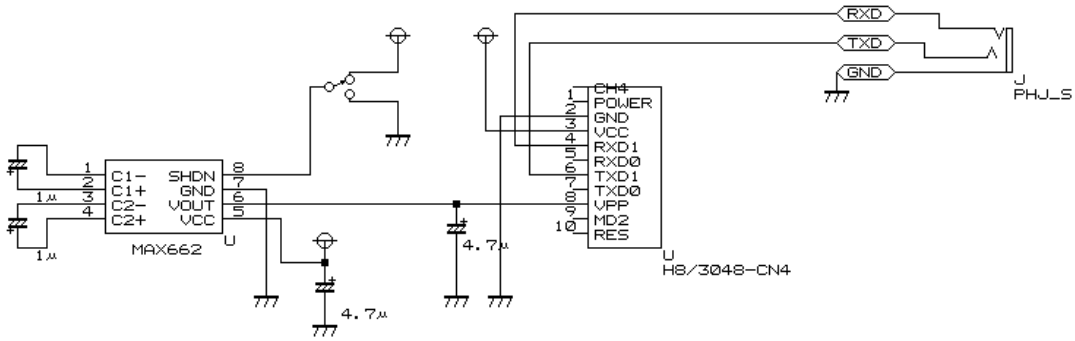
D-SUBコネクタ裏面より

3-6. フラッシュ ROM ライタボード

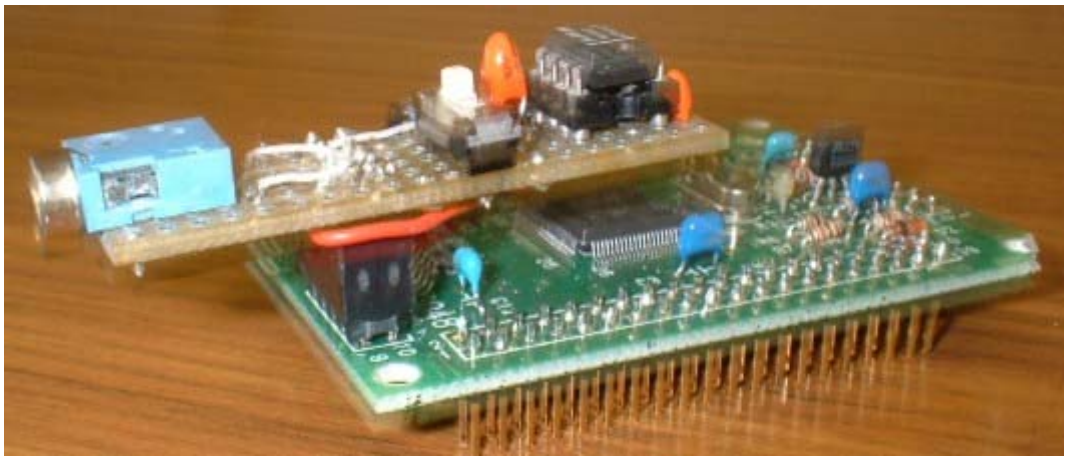


H8 内蔵のフラッシュ ROM への書き込みのためには電圧 12V が必要となるが、5V 単電源で書き込みができるようにするために、ステレオジャック（メス）が付いたフラッシュ ROM ライタボードを作る。基板上に直接取り付けてもよいが、汎用性を考えてアダプタ方式とする。アダプタを取り付けるため、AKI-H8 の足 CN4 を上向きに付けておく。

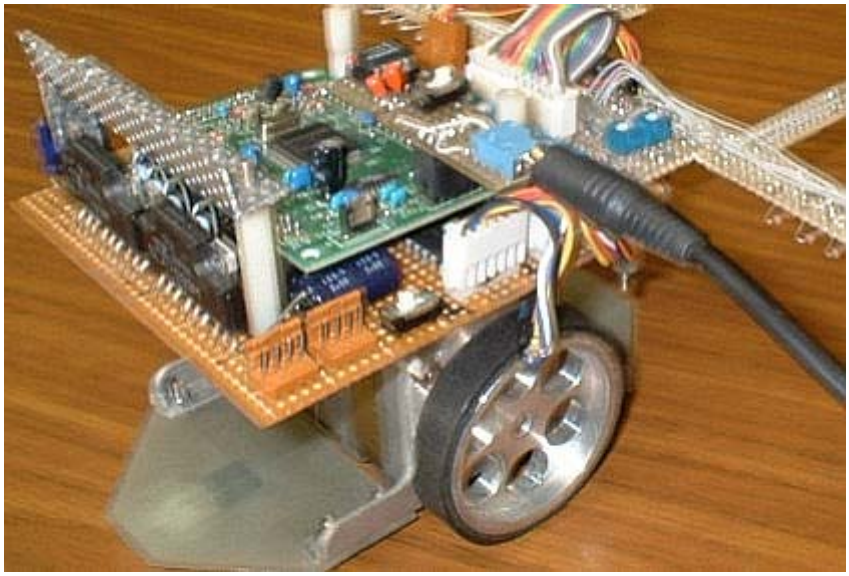




切り替えスイッチが HI のときが実行モード、LOW のときが書き込みモードである。プログラム転送をするときはスイッチを書き込みモードにし、実行するときは ROM ライタボードを取り外すか、スイッチを実行モードにする。



フラッシュ ROM ライタボードを AKI-H8 に挿したところ

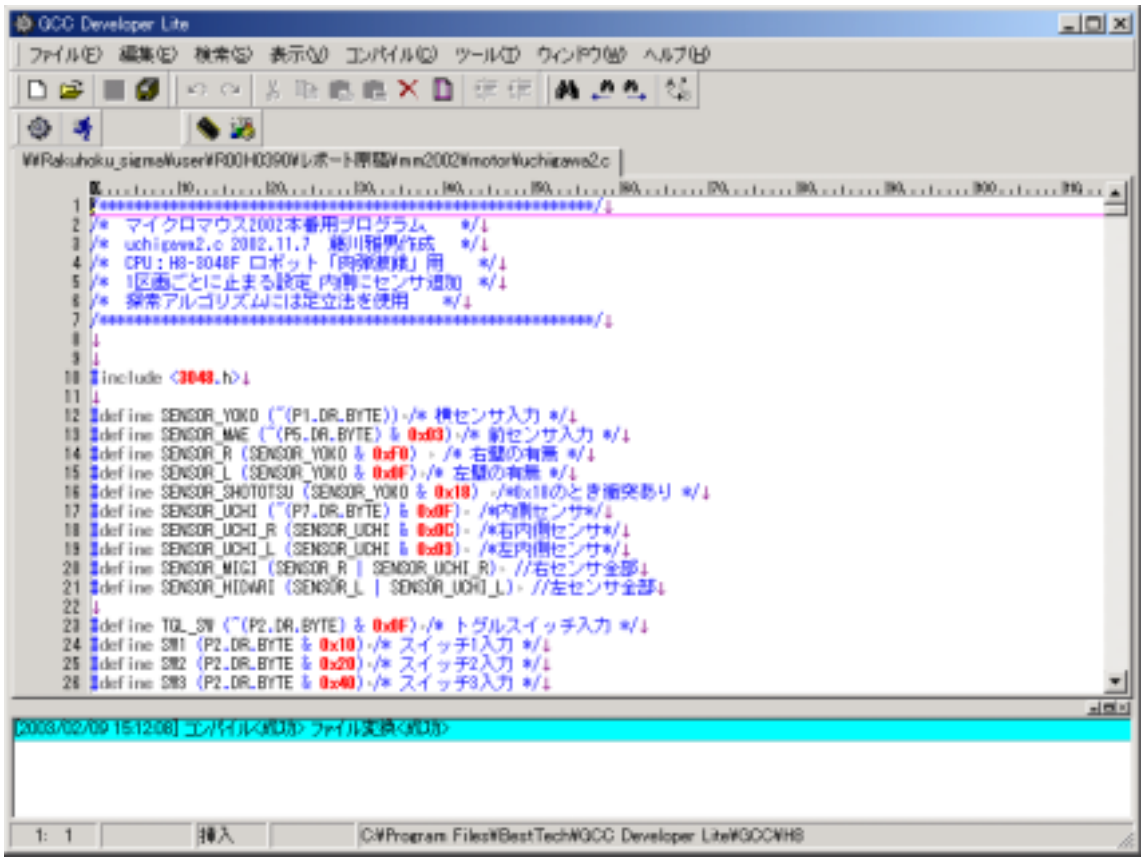


H8 をマザーボードに差し替える手間なしに転送を行うことができる。

3-7. GCC

H8 用の C 言語のコンパイラにはいろいろなものがあるが、今回はフリーのコンパイラ GCC を用いた。GCC を用いた開発環境として、ベストテクノロジー社の GCC Developer Lite がフリーであり、かつ使いやすかったのでこれを採用した。

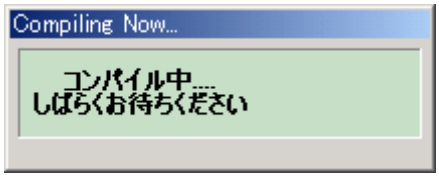
3-7-1. GCC Developer Lite



GCC Developer Lite 以外に GCC とそれを Windows 上で動作させるための Cygnus GNUPro Toolkit をインストールする必要があるが、これらは新しいバージョンの GCC Developer Lite のインストーラで自動的にインストールすることができる。

GCC Developer Lite はベストテクノロジー社のホームページで配布されている。

ベストテクノロジー社：http://www.besttechnology.co.jp/download/



コーディング・コンパイル・転送ソフト FW の呼び出しをすべてこのソフト上で行うことができる。3048 以外の H8 シリーズのコーディングも可能である。

詳しい使い方については付属のマニュアルを参照のこと。

3-7-2. GCC を使う場合の H8-3048 用 C プログラムの書き方

3-7-2-1. インクルードファイル

H8-3048 のプログラミングに便利なように「3048.h」というファイルが用意されているので、これを利用する。

具体的には、プログラムの頭に

```
#include <3048.h>
```

の行を追加する。これによって、レジスタへの書き込みのとき番地を書くのではなく、レジスタ名を使うことができるようになる。

（例）

```
ITU0.TCR.BYTE = ITU1.TCR.BYTE = 0x23;
ITU.TSTR.BYTE = 0x03;
ITU0.TIER.BYTE = ITU1.TIER.BYTE = 0x01;
```

3-7-2-2. 入出力命令

H8-3048F は I/O マップド I/O 方式ではなくメモリマップド I/O 方式であるため、IN 命令や OUT 命令は存在しない。

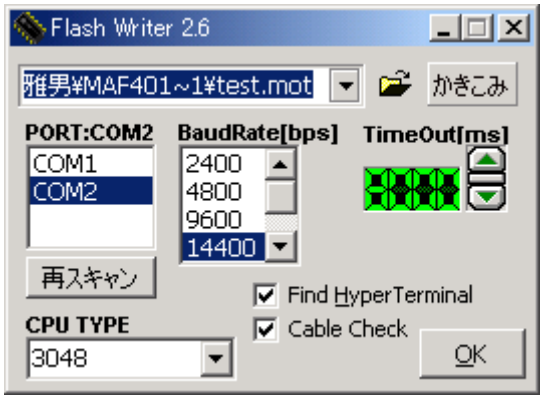
各ポートはビットごとに入力・出力の設定をすることができる。設定は各ポートのデータディレクションレジスタ（DDR）で行う。ここで 0 を書き込んだビットは入力、1 を書き込んだビットは出力になる。入出力はデータレジスタ（DR）を使って行う。

（例）

```
PA.DDR=0xFF;  /*ポート A 全ビット出力（モータ）*/
PA.DR.BYTE=0; /*ポート A から 0x00 を出力*/
PB.DDR=0xFF;  /*ポート B 全ビット出力（モニタ LED・ブザー）*/
PB.DR.BYTE=0; /*ポート A から 0x00 を出力*/
P1.DDR=0x00;  /*ポート 1 全ビット入力（センサ横）*/
P2.DDR=0x00;  /*ポート 2 全ビット入力（スイッチ）*/
P5.DDR=0x00;  /*ポート 5 全ビット入力（センサ前）*/
```

3-8. FW

コンパイルしたプログラムをマイコンのフラッシュメモリに書き込むためのソフトとして、フリーソフト Flash Writer を使った。この他、Windows 付属の「ハイパーターミナル」等でも転送は可能である。

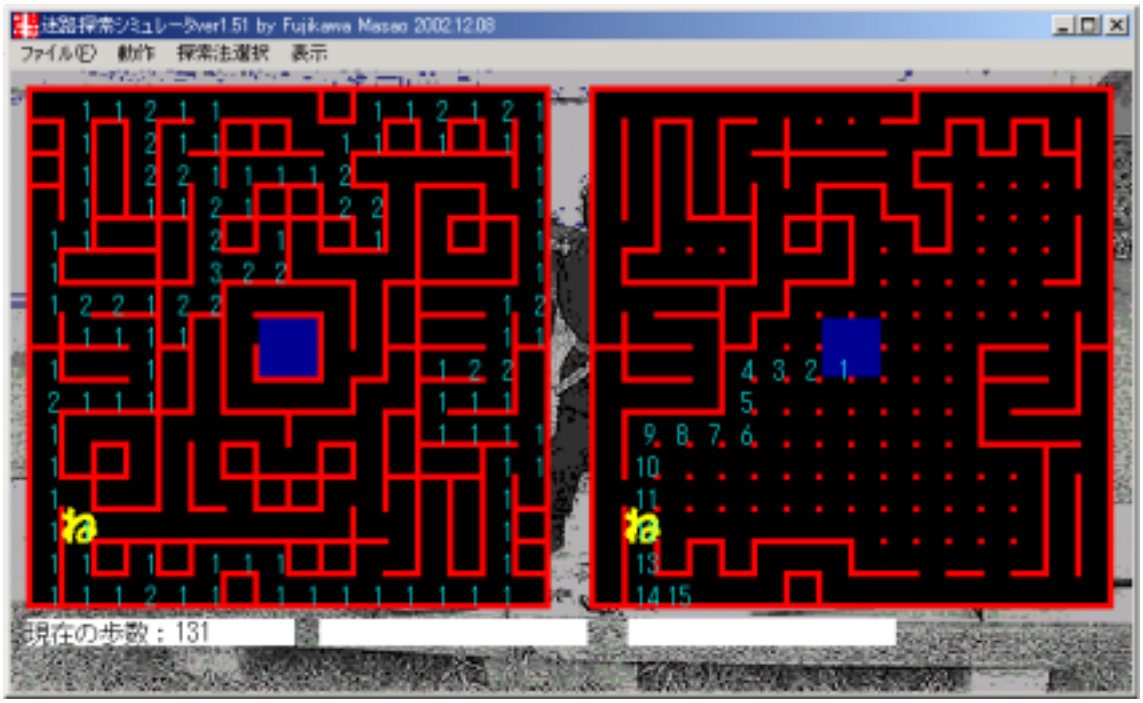


FW の使い方

ポート番号・ボーレート・CPU のタイプ・タイムアウトまでの時間を設定する。転送したいプログラムを選び、「かきこみ」ボタンをクリックする。転送に成功したら成功した旨が表示される。

ボーレートは 14400 か 9600 程度にする。うまくいかないときは遅くしてみる。タイムアウトの時間は短くしすぎないようにする。

3-9. 迷路探索シミュレータ



モータドライブやセンサをつくり、動作確認をするときは実際の回路を作成する必要がある。しかし、迷路探索はそれらに比べて抽象性が高いため、実際に回路を作らなくてもさまざまなアルゴリズムの動作確認をすることができる。むしろ、ハードウェア的な問題を考えなくてよいため、シミュレータ上のほうが好都合であるとさえいえる。

そのような考えから、自分で開発した迷路探索シミュレータを用いてアルゴリズムの研究を行った。ソースコードを書きなおして動作させることによって、そのアルゴリズムが有効かどうかを確認する。

実際の探索についてのソースは、robot.h 内の KetteiNextKodo 関数の中に次のように書く。

SensorData 構造体に、前後左右の壁のあるなしその他の情報（その場所に今まで何回来たことがあるか、など）があるため、それを参考に次の行動（ただの前進・90 度右回転して前進・90 度左回転して前進・180 度回転して前進）を決定し、返す。

プログラムの例

```
//単純左手法（左＞前＞右＞後の順に優先順位をつけて迷路をたどる方法）
case(TANJUNHIDARITEHO):
    if(SensorData.Hidari == 0)
        return (TURN_L);
    else{
        if(SensorData.Mae == 0)
            return (TURN_NON);
        else{
            if(SensorData.Migi == 0)
                return (TURN_R);
            else{
                WriteKaisuData((ReadIchi()).cell_i,(ReadIchi()).cell_j,(ReadRobotCellData((ReadIchi()).cell_i
                ,(ReadIchi()).cell_j)).Kitakaisu+1);
                return (TURN_U);
            }
        }
    }
    break;
```

3-10. ステッピングモータを回転させるプログラム

ステッピングモータを回転させるには、増幅用素子に 4 つのパルス列を送ってやればよい。今回とっている 2 相励磁方式の場合のタイミングチャートは次のとおりである。

励磁信号タイムチャート
2相励磁

clock	0	1	2	3	0	1
IN_A	H	L	L	H	H	L
$IN_{\overline{A}}$	L	H	H	L	L	H
IN_B	H	H	L	L	H	H
$IN_{\overline{B}}$	L	L	H	H	L	L

clock0 から 0,1,2,3,0,1,2,3,…の順にパルスを与えていけばモータは回転する。逆回転をさせるときは 0,3,2,1,0,3,2,1,…の順にパルスを与えてやればよい。

今、I/O の割り振りは以下のようになっている。

PA0：右モータ A 出力
PA1：右モータ B 出力
PA2：右モータ¥A 出力
PA3：右モータ¥B 出力
PA4：左モータ A 出力
PA5：左モータ B 出力
PA6：左モータ¥A 出力
PA7：左モータ¥B 出力

右モータと左モータは互いに向かい合わせに配置してあるので、ロボットを前進させようと思ったら、片方を逆回転させなければならない。

ゆえに、ロボットが前進するときのパルスの与え方は次のようになる。

clock	0	1	2	3
PA0	1	0	0	1
PA1	1	1	0	0
PA2	0	1	1	0
PA3	0	0	1	1
PA4	1	1	0	0
PA5	1	0	0	1
PA6	0	0	1	1
PA7	0	1	1	0
PAの値	0x33	0x96	oxCC	0x69

つまり、PA から 0x33,0x96,0xCC,0x69 を循環するように出力してやれば、ロボットは前進することになる。

これを C 言語で表すと以下のようになる。

```
/*前進*/
int Pattern[4] = {0x33,0x96,0xCC,0x69};
int i = 0;
while(1){
      PA.DR.BYTE = Pattern[i];
      i = (i + 1) % 4;
}
```

後退の場合は配列をたどる順序を反対にすればよい。

```
/*後退*/
int Pattern[4] = {0x33,0x96,0xCC,0x69};
int i = 0;
while(1){
      PA.DR.BYTE = Pattern[i];
      i = (i + 3) % 4;
}
```

右に回転する場合は、右のモータを逆回転させ、左のモータを普通に回転させる。
この場合、右のモータと左のモータに与えるパルスの配列を別々にしておいたほうがプログラムを単純にできる。

```
/*右回転*/
int PatternR[4] = {0x03,0x06,0x0C,0x09};
int PatternL[4] = {0x30,0x90,0xC0,0x60};
int i = 0;
int j = 0;

while(1){
      PA.DR.BYTE = PatternR[i] | PatternL[i];
      i = (i + 3) % 4;
      j = (j + 1) % 4;
}
```

左に回転させるときは右回転のときの反対である。

3-10-1　タイマ

上記の方法では、与えるパルスの変化が早すぎてモータを回転させることができない。モータには回転できるスピードの限界があって、これより速すぎたり遅すぎたりすると正常な動作ができなくなる。これを脱調という。

脱調を防ぐためには、タイマで適当な時間稼ぎをしてやる必要がある。その方法としては、意味もなく数を数えるなどしてソフト的にタイマを実現してやる方法と、H8 がはじめから持っているタイマ割り込みの機能を利用して行う方法とがある。今回は CPU の負担を減らすためタイマ割り込みを利用した。

```
/*前進の例（ソフトタイマを使う方法）*/
int Pattern[4] = {0x33,0x96,0xCC,0x69};
int i = 0;
int j;
while(1){
    PA.DR.BYTE = Pattern[i];
    i = (i + 1) % 4;
    for(j = 0; j < 30000; j++){;}       /*時間稼ぎのため意味もなく数を数える*/
}
```

3-10-2. H8 のタイマ割り込み機能（ITU）

タイマの中心は 16 ビットのカウンタである。カウンタは 5 本あり独立動作でき、CPU が動作しているシステムクロックごとにカウントアップしている。0xFFFF まで数えたら 0 に戻り繰り返しカウントアップする。1 つのカウンタに 2 つのジェネラルレジスタ計 10 本が連動している。ジェネラルレジスタはカウンタと比較され、一致すると割り込み要求を発生したり端子の状態を変更できるので次のような目的で使われる。

- インターバルタイマ：時間がきたら割り込み要求で知らせる
- ワンショットパルス出力：時間がきたら端子を一回だけ変更する
- トグル出力：時間がきたら出力端子を反転する

以上の機能はジェネラルレジスタに時間を設定しておく。

ほかに、PWM 出力の機能もある。

タイマ機能（1 チャネルにつき）

- クロック：φ、φ/2、φ/4、φ/8、外部（TCLKA、TCLKB、TCLKC、TCLKD）
- ジェネラルレジスタ：2 本（チャネル 3/4 のみ）

- 割り込み要求：コンペアマッチ/インプットキャプチャ/オーバーフロー
- DMA 転送要求：可
- パルス出力：2 本（ワンショット/トグル/PWM）
- 出力停止機能：あり
- パルス入力：2 本（立上がり/立下り/両エッジ）

クロックは 16MHz であり、周波数が高すぎるので、分周機能を利用する。φ/8 にすると、2MHz まで下がるので 16 ビットマイコンで使いやすい長さになる。

例えば 20 ミリ秒たつごとになにかイベントを起こそうと考えると、タイマ割り込みごとに 1 回数えるとして

320000÷8＝40000（回）

数えればよいことになる。

カウンタにある値を入れ、タイマ割り込みが入るたびにカウンタの値を 1 つ減らすことにする。カウンタが 0 になったらパルスを 1 つ出す。これを繰り返すことによってモータを回転させることができる。

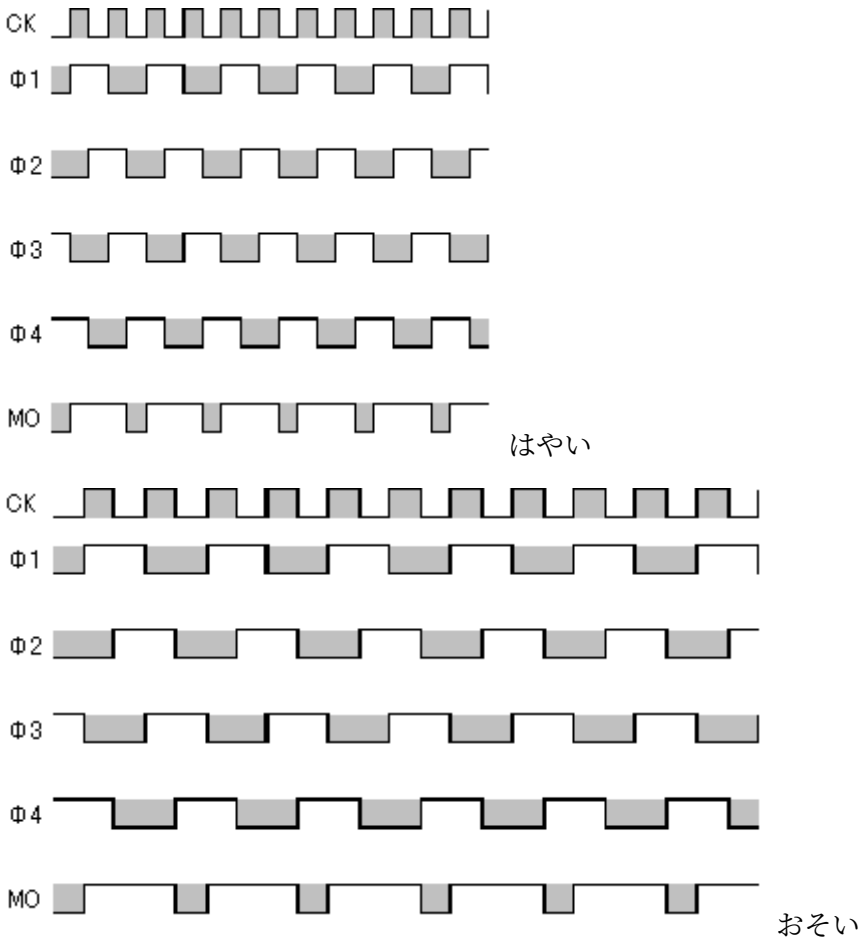
加速のさいはカウンタに入れる値を順次減らしていき、減速のときは逆に増やしていく。等速度運転のときは値を変えなければよい。

モータが対応できるパルスの周波数は、それぞれの製品によって異なる。高すぎても、低すぎても脱調する。

```
/*タイマ 0 の割り込み関数　割り込みで前進をする*/
int Ri;
void int_imia0(void)
{
    ITU0.GRA = 10000;   /*タイマセット*/
    /*モータ励磁パターンを一つ進める*/
    Ri = (Ri + 1) % 4;
    PA.DR.BYTE = PatternR[Ri]; /*割り込みパターン出力*/
        ITU0.TSR.BIT.IMFA = 0;   /*割り込み発生フラグをクリア*/
}
```

3-10-3. モータの速さを変える

モータの回転速度を変えるには、タイマが数える数を変えてやればよい。数える数を大きくするとパルスとパルスの間のインターバルが大きくなり、回転速度は落ちる。逆に、カウンタの値を小さくすると回転速度は上がる。



3-10-4. 左右のモータの速さを独立に変える

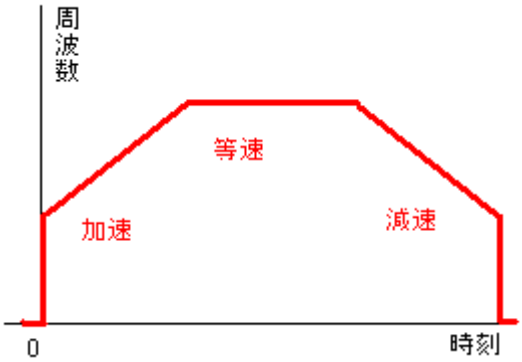
左右のモータのうち片方だけを少し遅くしてやることができれば、滑らかな軌道修正を行うことができる。

具体的なプログラムは台形加減速の説明の後に載せる。

3-11. 台形加減速

ステッピングモータを高速で駆動する場合、いきなり高い周波数を与えても脱調してしまう。そのため、スローアップ・スローダウンの制御方法が用いられる。これには直線形、指数関数型、S 字型スローアップ・スローダウンなどがある。スローアップとはモータが入力パルスに同期して回転するように、駆動周波数に適度な傾斜をもたせて加速することである。スローダウンは同じく減速することである。

今回はやさしい直線型を用いた。スローアップし、しばらく一定速度で回転してからスローダウンするというスピードの変化が台形をしているので台形加減速とも呼ばれる。実際は完全な台形ではなく、台形の両端を切り取った形をしている。



速度（周波数）を直線的に上げるためには、パルス周期を適切に決める必要がある。しかし、計算が面倒なのであらかじめパソコンなどで計算を済ませておき、テーブルとして持っておく方がよい。

左右独立に加減速を行うプログラムの例：

```
/*ITU0・ITU1 設定*/
ITU0.TCR.BYTE = ITU1.TCR.BYTE = 0x23;    /* GRA ｺﾝﾍﾟｱﾏｯﾁ  clock 1/8 */
ITU.TSTR.BYTE = 0x03;

/*  割り込みイネーブル  */
ITU0.TIER.BYTE = ITU1.TIER.BYTE = 0x01;

int table_i0 = table_i1 = 0;
/*加速のためのタイムテーブル*/
static int accel_tbl[] =
{
    9751,9600,9456,9319,9187,9061,8940,8823,8711,8603,
```

```
    8499,8399,8302,8209,8118,8031,7946,7864,7784,7707,
    7632,7559,7488,7419,7352,7287,7224,7162,7101,7043,
    6985,6929,6875,6821,6769,6718,6668,6619,6572,6525,
    6479,6434,6391,6348,6306,6264,6224,6184,6145,6107,
    6069,6033,5996,5961,5926,5892,5858,5825,5792,5760,
};
/*タイマ 0 の割り込み関数*/
void int_imia0(void)
{
  switch(Mot_mode){
    case ZENSHIN:
    case HIDARI_KAITEN:
    case HIDARI_SHUSEI:
    /*モータ励磁パターンを一つ進める*/
        Ri = (Ri + 1) % 4; break;
    case KOUTAI:
    case MIGI_KAITEN:
        Ri = (Ri + 3) % 4; break;
    case STOP:
        Ri = Ri;   break;
  }
  MOT_DR = PatternR[Ri] | PatternL[Li]; /*割り込みパターン出力*/
  if(Kagen_flg == KASOKU)   table_i0++;   /* 加速なら早くする  */
  else if(Kagen_flg == GENSOKU) table_i0--;   /* 減速なら遅くする  */
  else {;}/* 定速なら何もしない  */

  /* のこりパルス数を一つ減らす  */
  Nokori_pulse--;
  ITU0.GRA = accel_tbl[table_i0]; /*次のタイマセット*/
  ITU0.TSR.BIT.IMFA = 0;   /*割り込み発生フラグをクリア*/
}

  /*タイマ 1 の割り込み関数*/
void int_imia1(void)
{
  switch(Mot_mode){
```

```
        case ZENSHIN:
        case MIGI_KAITEN:
        case MIGI_SHUSEI:
        /*モータ励磁パターンを一つ進める*/
            Li = (Li + 1) % 4; break;
        case KOUTAI:
        case HIDARI_KAITEN:
            Li = (Li + 3) % 4; break;
        case STOP:
            Li = Li;    break;
    }
    MOT_DR = PatternR[Ri] | PatternL[Li]; /*割り込みパターン出力*/
    if(Kagen_flg == KASOKU)   table_i1++;   /*  加速なら早くする  */
    else if(Kagen_flg == GENSOKU) table_i1--;   /*  減速なら遅くする  */
    else {;}/*  定速なら何もしない  */

    /*  のこりパルス数を一つ減らす  */
    Nokori_pulse--;
    ITU1.GRA = accel_tbl[table_i1]; /*次のタイマセット*/
    ITU1.TSR.BIT.IMFA = 0;   /*割り込み発生フラグをクリア*/
}
```

3-12. センサから情報を受け取るプログラム

センサから情報を受け取るには、ポートの DR を見ればよい。

（例）

```
P1.DDR=0x00;   /*ポート 1 全ビット入力（センサ横）*/
if(P1.DR != 0){……}
```

3-13. 一区画前進と旋回

マイクロマウスの基本動作は次の四つである。

- 1 区画走行
- 90 度右回転
- 90 度左回転
- 180 度回転

3-13-1. 必要パルス数の計算

まず理論値を求め、その後に実際の走行にあわせて値を加減する。

3-13-1-1. 車体の理論値

モータ回転：200 パルス/一周

タイヤ直径：50mm

∴タイヤ周：50πmm

∴パルス一発で進む距離：π/4mm

車輪中心から車輪中心：92mm

∴極地旋回一周：92πmm

∴右回転・左回転：92π/4=23πmm（92 パルス）

180 度回転：46πmm（184 パルス）

3-13-1-2. 迷路の理論値

柱の太さ：12mm

柱と柱の間：168mm

一区画：180mm

∴一区画の距離：180/(π/4)パルス分（229 パルス）

3-13-1. 一区画前進

台形加減速を用いる。

50 パルスで加速後、147 パルス定速前進し、32 パルスで減速という設定にした。前センサと車体の間の距離が短かった都合上、減速のパルス数を少なくしている。

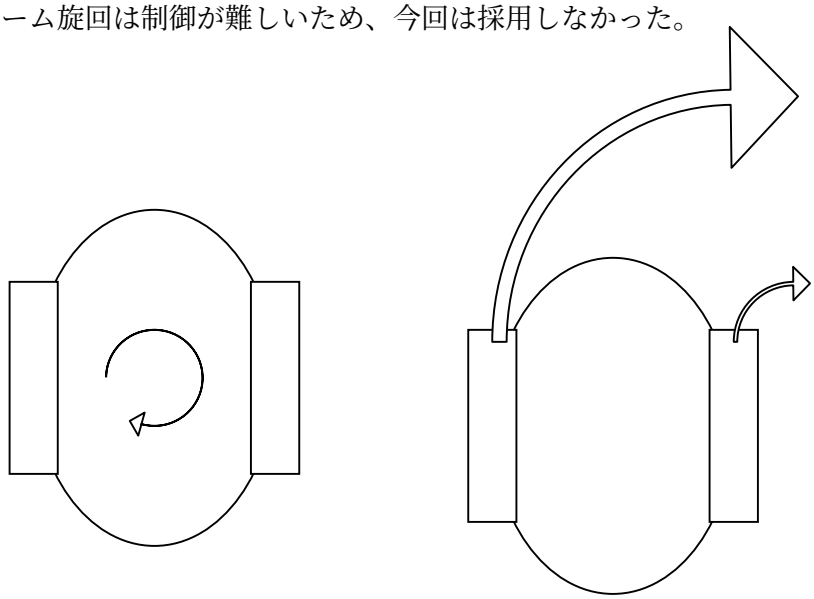
3-13-2. 旋回

旋回には、極地旋回とスラローム旋回がある。

極地旋回とは、左右の車輪を逆方向に回転させ、その場で旋回する方法である。これに

対しスラローム旋回とは、左右の車輪の回転スピードを変えて角を曲がる方法である。一旦停止や逆回転をせず、常に前進を続けられるので極地旋回より高速である。

スラローム旋回は制御が難しいため、今回は採用しなかった。



スラローム旋回

極地旋回でも台形加減速を用いている。90 度回転のとき加速 20 パルス、定速 52 パルス、減速 20 パルス。180 度回転のときは加速 30 パルス、定速 124 パルス、減速 30 パルスとしている。回転のスピードが速すぎると回りすぎるので注意が必要である。

3-14. 姿勢制御

迷路の１区画の中央と、ロボットの回転中心が一区画前進のたびに一致するようにすることで、正確な制御を行うことができる。姿勢制御は、左右の位置と前後の位置の両方で行う必要がある。前進のときだけ行い、90 度回転や 180 度回転のときは行わない。

必要なデータを整理して理論値を求めると次のようになる。

前センサから車体中心：115mm

前センサから横センサ：40mm

横センサから車体中心：75mm

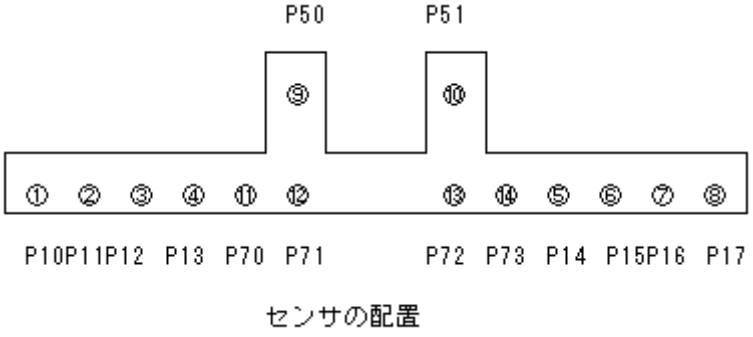
∴切れ目検出から次の区画の中心：84+75=159mm（202 発）

前センサ検出から次の区間の中心：115-90=25mm（32 発）

衝突センサ（左右センサの一番内側が同時についた状態）検出からその区間の中心：90-75=15mm（19 パルスバック）

壁の切れ目から 159 ミリ（202 パルス）で車体中心と迷路中心が一致。

1 区画 229 パルスだからスタートしてから 27 パルスで切れ目が来る。



センサの配置

3-14-1. 左右の姿勢制御

ロボットが迷路の幅のちょうど中央に位置しているとき、左壁はセンサ②と③の間に、右壁は線さ⑥と⑦の間に来るようになっている。ここから外れるとロボットは左右いずれかに寄りすぎであると判定し、修正動作を行う。修正動作は、一方の車輪を遅くすることによって行う。なお、左右どちらの壁を見て修正を行うかは以下のとおりである。

１．右壁があったら、右壁センサで修正

２．右壁がなくて左壁があれば、左壁センサで修正

３．両方壁がなければ修正しない

当初、①～⑧のセンサしかなかったところに⑪～⑭のセンサを追加したので、場合わけがすこし複雑になっている。

場合わけ

P7（内側）のセンサが反応していないとき

横センサ配置は左から　P10 P11 P12 P13 P14 P15 P16 P17　であるから、P1 からの入力値およびすべき動作は以下のようになる。

左センサ	入力値	右センサ	入力値	意味	すべき動作
XOOX	0x06	XOOX	0x60	中央	何もしない
XOXX	0x02	XOXX	0x20	右+	少し左遅く
OOXX	0x03	OOXX	0x30	右++	左遅く
OXXX	0x01	OXXX	0x10	右+++	さらに左遅く
XXXX	0x00	XXXX	0x00	はずれ	何もしない
XXOX	0x04	XXOX	0x40	左+	少し右遅く
XXOO	0x0C	XXOO	0xC0	左++	右遅く
XXXO	0x08	XXXO	0x80	左+++	さらに右遅く

P7（内側）のセンサが反応しているとき

内側のセンサが反応しているときは、反応していないときの大修正よりも大きな修正を行う。また、迷路の大きさより、内側のセンサは右と左が同時に動作することはない。

左センサ	入力値	右センサ	入力値	意味
XO	0x02			左++++
OO	0x03			左+++++
OX	0x01			左++++++
		XO	0x08	右++++
		OO	0x0C	右+++++
		OX	0x04	右++++++

3-14-2. 前後の姿勢制御

前後の姿勢制御のためには、前の壁を利用する方法、前の壁との正面衝突を利用する方法と、左右の壁の切れ目を利用する方法とがある。

3-14-2-1. 前の壁を利用する方法

前の壁をセンサが感知してから 32 パルスで止まれば、ちょうどよい位置にロボットが来るはずである。しかし、さまざまな要因によりずれが生じるので、これ以外の方法も用いる。

3-14-2-2. 正面衝突時前後位置補正

正面衝突検知の仕組みとしては「右内側のセンサと左内側のセンサが同時に反応したときは正面衝突したものとみなす」という考え方を用いた。この方法だとその他のぶつかり方をしたときの修正には対応できないが、実験の結果ぶつかるときは大体正面衝突であることがわかったので、これに限って追加することとした。

3-14-2-3. 壁の切れ目を検出して前後位置補正

左右の壁の切れ目を利用して前後位置補正を行う方法である。壁の切れ目とは、左あるいは右のセンサの反応が

壁あり→壁なし

あるいは

壁なし→壁あり

に変化する点である。これを検出する方法は、一つ前のセンサの反応を保存しておき、現在の反応と比べるというものである。

3-14-3. 軌道修正を加えた一区画前進

もともと、横壁情報は定速度運転の時にしか取らなかった。また、保管されている壁情報が 1 区画ごとにクリアされるので、加速途中に前の壁に正面衝突したときマウスは次にどのような動作をするべきかわからず、擱挫してしまうことになった。この問題の解決法の一つとして、正面衝突したときはそこから迷路の中央位置にまで後退し、そこで横壁情報を改めてとり、つぎの動作を行わせるということをおこなった。

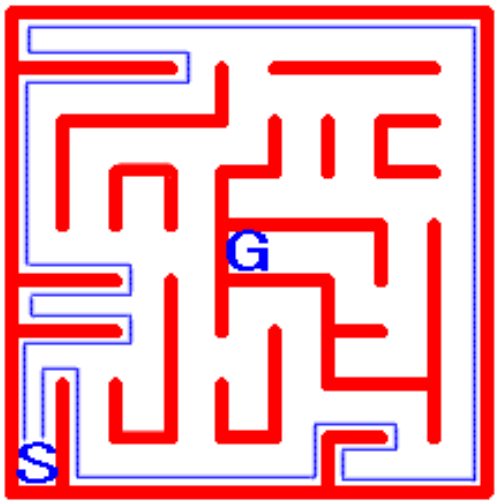
軌道修正を加えた一区画前進を設定すると次のようになる。

①50 パルス加速（途中壁の切れ目を検出したら残りパルス数を 23（∵50-27）にする）
②66 パルス定速（途中壁の切れ目を検出したら残りパルス数を 89（∵66+23）にする）
③一回目の横壁データサンプリング
④20 パルス定速
⑤二回目の横壁データサンプリング
⑥61 パルス定速（途中前壁検出したらすぐに減速に移る）
⑦32 発で減速（途中前壁検出したら残りパルス数を 32 に戻す）

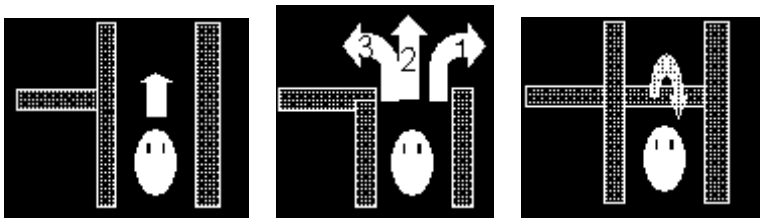
3-15. 迷路探索

迷路探索の方法には、右手法・求心法・足立法といったものがある。

3-15-1. 単純な右手法



単純な右手法では、次のような条件にしたがって次に進む方向を決定する。



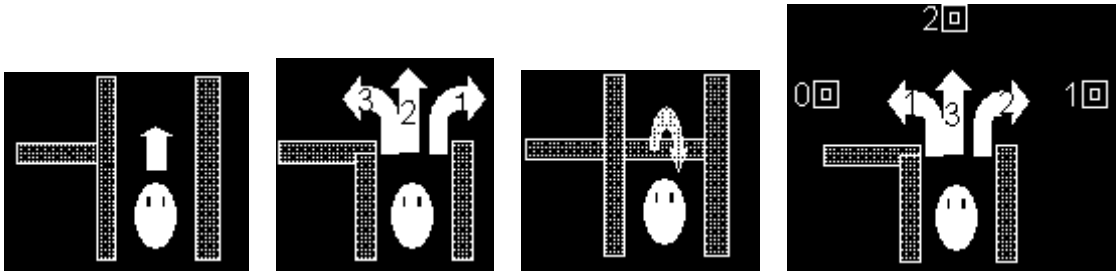
条件１：左右に壁があるときは直進する。
条件２：分岐点にきたときは、右・前・左の順に優先順位をつける。
条件３：袋小路は U ターンする。

スタートとゴールが壁でつながっている迷路ならこれで解けるが、マイクロマウスの迷路では中央にゴールがありスタートとゴールが壁でつながっていないため、無限ループに陥り解くことができない。

優先順位を左・前・右の順にしたものは左手法とよばれる。

3-15-2. 拡張右手法（来た回数を用いる方法）

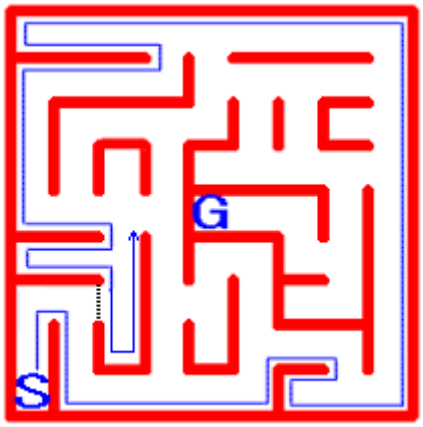
単純な右手法を拡張した方法である。2001 年大会の時はこのアルゴリズムを用いて完走した。



条件１：左右に壁があるときは直進する。
条件２：初めての分岐点にきたときは、右・前・左の順に優先順位をつける。（左手法は逆）
条件３：1 回以上通過したことのある分岐点にきたときは通過回数の少ないほうへ進む。通過回数がすべて同じであった場合には条件２による。
条件４：袋小路は U ターンする。

3-15-3. 拡張右手法（仮想壁を用いる方法）

同じく単純な右手法を拡張した方法である。実際には壁はないが、仮にあることにして無限ループに陥るのを回避する。



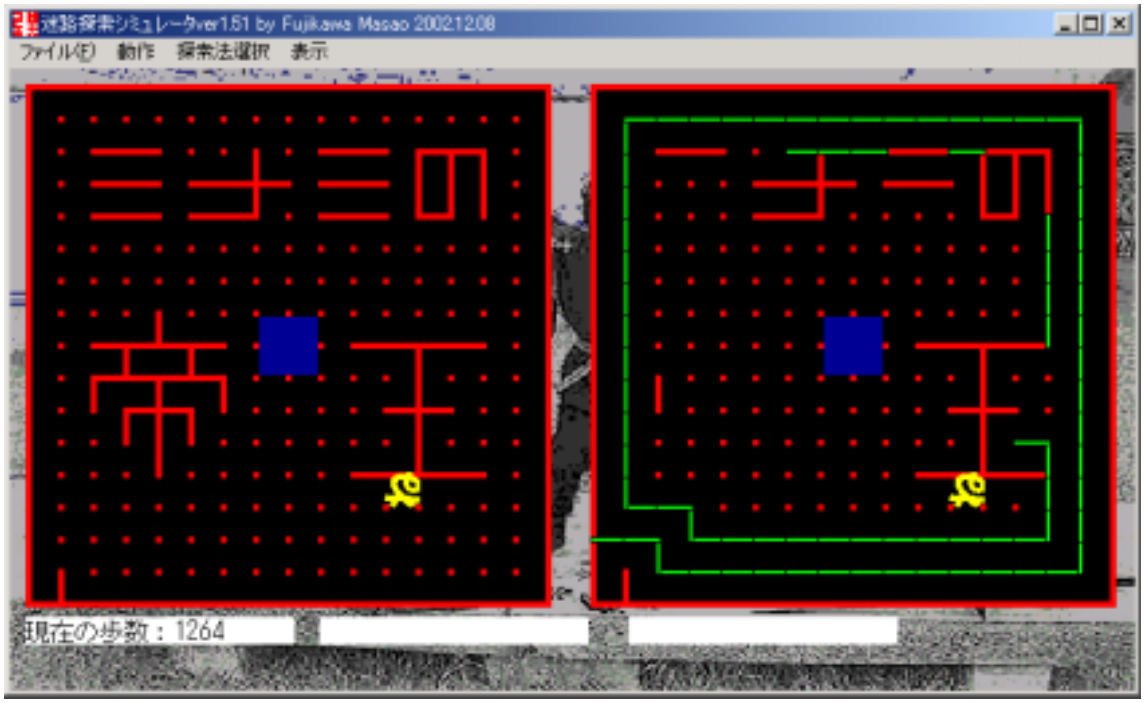
仮想壁を使うアルゴリズムを C 言語風にあらわすと以下のようになる。

```
while(1){
  if(右に行ける  and  右の区画に行ったことがない){
      右に行く
  }else{
      右に仮想壁を書く
      if(前に行ける  and  前の区画に行ったことがない){
          前に行く
```

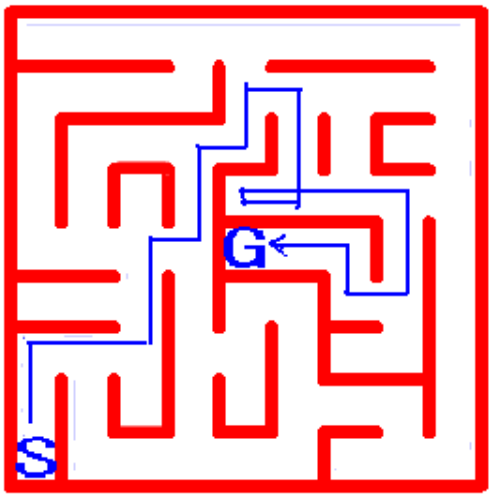
```
        }else{
            前に仮想壁を書く
            if(左に行ける and 左の区画に行ったことがない){
                左に行く
            }else{
                左に仮想壁を書く
                行ったことのない区画にたどり着くまで通常の右手法で移動する（仮想壁も通常の壁と同じように扱う）
            }
        }
    }
}
```

3-15-4. 仮想壁を用いる方法が通用しない場合

迷路によっては、仮想壁を用いる方法では解けない場合もある。



3-15-4. 求心法



迷路の中央（ゴール）を目指して進みつづける方法である。

右手法（左手法）より 1 回目は早く着くが、全面探索には向かない。

迷路の各区画にたとえば以下のような「重み」を与える。ゴールがもっとも重みが大きくなる。

0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1
2	3	4	5	6	7	8	9	9	8	7	6	5	4	3	2
3	4	5	6	7	8	9	10	10	9	8	7	6	5	4	3
4	5	6	7	8	9	10	11	11	10	9	8	7	6	5	4
5	6	7	8	9	10	11	12	12	11	10	9	8	7	6	5
6	7	8	9	10	11	12	13	13	12	11	10	9	8	7	6
7	8	9	10	11	12	13	14	14	13	12	11	10	9	8	7
7	8	9	10	11	12	13	14	14	13	12	11	10	9	8	7
6	7	8	9	10	11	12	13	13	12	11	10	9	8	7	6
5	6	7	8	9	10	11	12	12	11	10	9	8	7	6	5
4	5	6	7	8	9	10	11	11	10	9	8	7	6	5	4
3	4	5	6	7	8	9	10	10	9	8	7	6	5	4	3
2	3	4	5	6	7	8	9	9	8	7	6	5	4	3	2
1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1
0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0

分岐点に来たときは重みが大きいほうに進む。

これも来た回数のカウントや仮想壁により無限ループに陥るのを防ぐ必要がある。

3-15-5. その他の方法

トレモー法

数学者トレモーによって考案された方法で、迷路のすべての区画を回るのに適した方法である。

条件１：左右とも壁がある場合は直進する。

条件２：未通過の角ではどの方向に進んでもよい（たとえば中央を目指すことにする）。

条件３：新しい道を通って通過済の角にきた場合は U ターンする。

条件４：通過済の道を通って角へきた場合は通過回数の少ない方に進む（優先順位右・前・左）。

条件５：袋小路では U ターンする。

足立法

足立法とは、マウスが知らない壁を仮に「ない」ものとして最短経路を求め、それにしたがって前進し、もし壁に阻まれたら新たに計算しなおすということを繰り返す方法である。足立というのは考案者の名前である。最短経路を考えながら進むので、ほかの方法（ゴールに着いてから最短経路を考える）に比べて合理的な進み方ができる。足立法については最短経路計算の後に詳説する。

3-16. 最短経路計算

マイクロマウス大会では五回のトライアルが許されている。たとえ一回目のゴールに長い時間がかかったとしても、二回目のゴールが早ければ問題ないといえる。二回目以降のスピードアップを図るためには「とにかくゴールに着く」だけでは不十分であり、覚えた迷路をもとに最短経路の計算をしてやる必要がある。

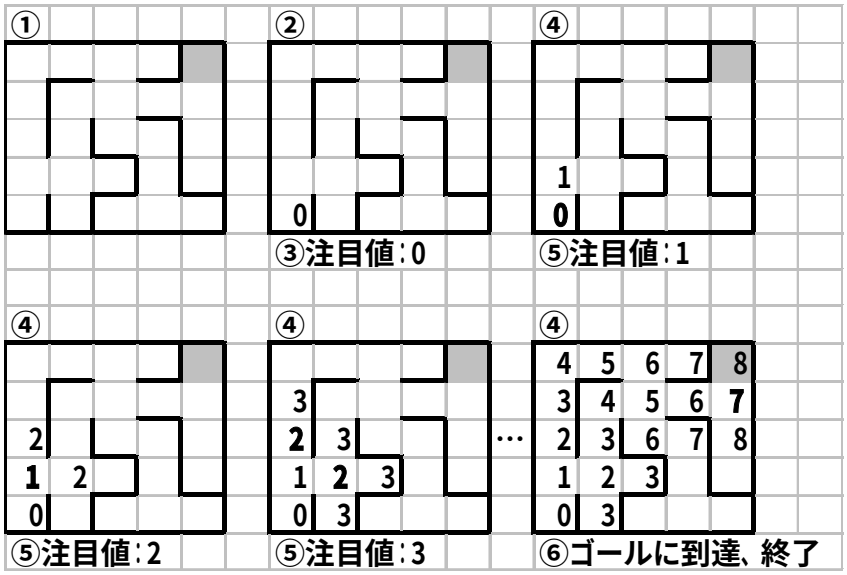
3-16-1. 歩数マップ

最短経路を求める方法として、歩数マップ（＝等高線マップ）を用いた方法がある。

隣接する区画に動く動作を一回の移動とする。歩数とは、スタートからある区画まで何回の移動で到達できるかを表したものである。

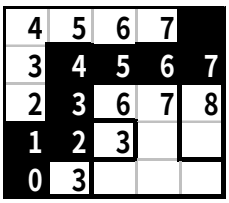
歩数マップの作成の方法は以下のとおりである。

① はじめは、歩数マップには何も書かれていないとする。
② スタート地点の歩数を 0 とする。
③ 注目する値を 0 とする。
④ 注目する値を歩数として持つ区画を探す。そこから一回で移動でき、かつ歩数を書き込まれていない区画に（注目する値＋１）を書き込む。
⑤ 注目する値を 1 増やす。
⑥ ④⑤を繰り返し、ゴールに歩数が書き込まれたら歩数マップの作成を終了する。



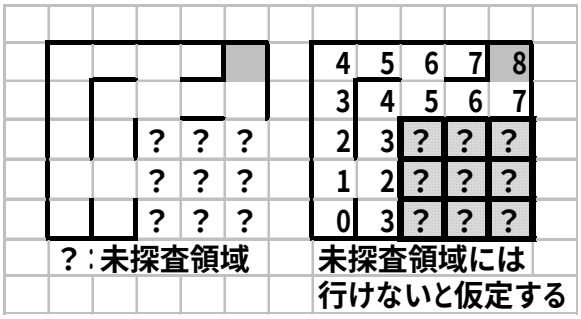
3-16-2. 最短経路

歩数マップを見て、ゴールから 1 ずつ少ない数字の区画をたどっていくと、最短経路が得られる。



最短経路が 2 つ以上発見されるときもある。その場合、「直線コースの多いほうを優先する」、「くし形経路はなるべく避ける」など、マシンの得手・不得手に応じた優先順位を決めておき、どちらの経路をたどるか判定することになる。

3-16-3. 未探査領域のあつかい



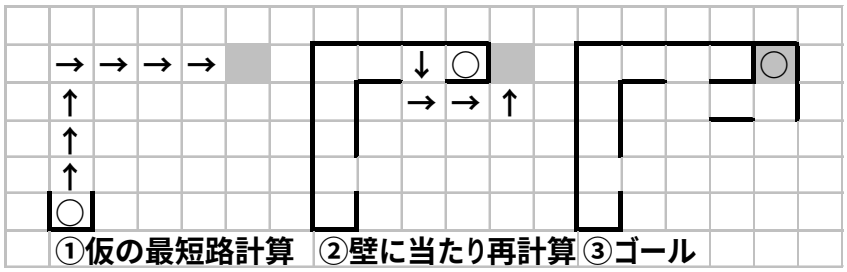
通常、歩数マップを作成するとき、「未探査領域には行けない」ものと仮定する。未探査領域には何があるかわからないので、確実なゴールまでの最短経路を求めるためにはそうしなければならない。そのため、いったんゴールに入るまでは最短経路の計算はしない。

3-16-4. 足立法

3-16-4-1. 足立法とは

まず迷路探索をしてゴールインしてから最短経路を求めるというのが、通常の考え方である。足立法はそうではなく、はじめから最短経路を計算しながらゴールを目指すという方法である。

真の最短経路の計算は、一旦ゴールしてからでないとできない。そのため、ここでは「**未探査領域には壁がない**」
ものと仮定して、スタートからではなく**現在位置**からゴールまでの仮の最短経路を計算する。そして、それにしたがって走行し、予期しなかった壁を発見して進めなくなった場合は計算をしなおして経路を修正する。これを繰り返せばゴールに到達することができる。



足立法が優れている点は、次の二つである。

- 迷路の隅のほうにあることが多い袋小路に入ることが少ない。そのため、無駄な動きで電力を消耗しなくてすむ。
- 迷路探索と最短経路計算を同時に行うことができる。

3-16-4-2. 足立法とその他の探索方法の比較

シミュレータで 2000 年・2001 年エキスパートコースの迷路を探索してみたところ、一回目のゴールまでかかった歩数として以下の結果を得た。

単位：歩

	拡張右 (回)	拡張左 (回)	拡張右 (壁)	拡張左 (壁)	求心	トレモー	足立
00 年 ex1	984	768	218	258	242	310	216
00 年 ex2	238	358	330	ゴール不能	95	95	85
01 年 ex1	454	186	288	298	64	64	92
01 年 ex2	272	311	ゴール不能	255	157	149	88

（注：現在のエキスパートコースルールでは反則となる動作（ゴール前にスタートに戻るなど）も含まれているので、この数字はあくまでも目安である。）

ここから、次のことがわかる。

- 拡張右手法（左手法）の回数による探索方法は、時間と電力を食いすぎるため使い道にならない。
- 拡張右手法（左手法）の仮想壁による探索方法は、回数による方法よりはましだが、ゴール不能になる可能性もあり、安全ではない。
- 求心法・トレモー法の場合ゴールは早いが、迷路によってむらがある。

その他の迷路での実験もふまえ、経験的に足立法が一番安定しているように思われた。そこで、大会のときは足立法を採用した。

４．考察など

4-1. マイクロマウス 2002 の顛末

2002 年 11 月 9 日、マイクロマウス 2002 エキスパートクラスで出場。

当初足立法のアルゴリズムどおりに進んでいったが、２分を過ぎたあたりから異常が発生。あきらかに異常な動作となり、リタイヤする。２回目以降ははじめから異常動作で、５回目まで同様にゴールすることができなかった。

4-2. 異常動作の原因

もともとセンサは左右 4 個ずつだったものを、より精度の高い軌道修正を行うために大会直前になって左右 6 個ずつに改造した。センサの発光ダイオードには、感度を得るためあまり大きな抵抗をつけていない。また、常時点灯のため、大電流が流れつづける。そのため、定格１アンペアを越える電流が流れつづけ、三端子レギュレータを破壊した。結果として電圧異常が起こり、CPU の異常動作を引き起こした。

三端子レギュレータの異常を回避する方法としては、以下のようなことが考えられる。

- センサの数を減らす。
- 常時点灯をやめ、パルス点灯を採用する。
- より大きな負荷に耐えられる三端子レギュレータに換装する。

負荷が定格を超えることを分かっていながら、「冷やせばなんとかなるだろう」という甘い見通しで対応を怠ったことが今回の敗因であった。

4-3. 今後の課題

マイクロマウス 2001 の反省として、以下のことを目標としてあげた。

●正確な探索を行えるようにする。

●最短走行経路を計算できるようにする。

そのうえで、

●高速な動きを可能とするハードウェアを開発する。

●ほかの CPU を使う（H8 など)、他の言語（C など）を用いる等、技術についての知見を広げる。

これらの目標のうち、ソフトウェア上の課題はとりあえずクリアしたが、それに対応できる機体を準備できなかったのが失敗の原因である。電源の欠陥以外にも、「左右のバランスが悪く、修正にかなり頼らないと走行できない」などさまざまな問題があった。

今後はハードウェアに重点を置き、まず現在のソフトウェアに完全に対応できるマシンを開発した上で「スラローム走行」「斜め走行」などのより高度な走行に挑戦していきたい。

５．まとめと謝辞

久保田先生、ならびに大会出場にご協力いただいた制御通信部の皆様に感謝いたします。

６．付録

全日本マイクロマウス大会競技規則（抜粋）

１－１　マイクロマウスは自立型でなければならない。

１－２　マイクロマウスは、競技中に操作者により、ハードウェアおよびソフトウェアの追加、取りはずし、交換、変更を受けてはならない。

１－４　マイクロマウスは迷路の壁を飛び越し、よじのぼり、傷つけ、あるいは壊してはならない。

１－５　マイクロマウスの大きさは、その床面への投影が１辺２５ｃｍの正方形に収まらなければならない。走行中に形状が変化する場合も、常にこの制限を満たしていなければならない。ただし、高さの制限はない。

２－１　迷路の壁の側面は白、壁の上面は赤、床面は黒とする。

２－２　迷路は１８ｃｍ×１８ｃｍの単位区画から構成され、全体の大きさは１６×１６区画とする。区画の壁の高さは５ｃｍ、厚さは１．２ｃｍとする。

２－３　迷路の始点は、四隅のいずれかにあり、時計回りに出発する。終点は中央の４区画とする。

３－１　マイクロマウスが始点から終点への走行に要した最短の時間をそのマイクロマウスの迷路通過時間記録とする。

３－２　操作者は迷路が公開された後で迷路に関する情報をマイクロマウスに入力してはならない。

３－３　迷路の走行は、毎回、始点より開始し、マイクロマウスが走行中止あるいは始点に戻った時点で終了する。

３－６　マイクロマウスは７分間の持時間を有し、この間５回までの走行をすることができる。

参考文献

「やさしいマイコン制御ロボットの製作」（横山直隆著、シータスク）
「ロボティクス入門」（ロボティクス研究会編、コロナ社）
「初心者のためのロボットの作り方第六版」（マイクロマウス委員会東日本支部）
「高速マイクロマウスの作り方」（浅野健一著、東京電機大学出版局）
「H8 ビギナーズガイド」（白土義男著、東京電機大学出版局）
「マイコン技術教科書 H8 編」（今野金顕著、CQ 出版）
「H8 マイコン完全マニュアル」（藤沢幸穂著、オーム社）ほか