e - puck を用いた自己位置推定システムに関する研究

コース　基盤科学コース
学籍番号　０８０５７９
氏名　東直人
指導教員　ルジェロ・ミケレット

Recently there is increasing interest in the realization of robots that are able to move independently from human control. For this reason I realized an original software for a small-sized robot called e-puck. This robot mounts several distance sensors and actuators. I realized an algorithm that is able to determine the position of the robot with good accuracy and automatically using an odometry calculation that is thesis I present in details the realization of the original system and the tests on the e-puck robot.

１．研究背景と目的

近年、人間の操作を必要としない自分で考え行動する自律移動ロボットへの期待が高まっている。現在では、数多くの企業・大学・研究所で研究がなされている。自律移動には、ロボットが周囲の環境変化を正確に認識し、ロボットが周囲の対象物の動作を予測する必要がある。その為には、ロボット自身の自己位置推定の機能が必要不可欠となってくる。自己位置推定法として、予めランドマークを設置しておくことや GPS 通信機能を利用することなどの手法がある。しかし、それらは既知環境や遮蔽物のない屋外環境でしか機能せず汎用性が低い。そこで、我々は未知環境や屋内で機能する自己位置推定システムの開発を目指している。具体的に、自己位置推定の古典的手法である Odometry を小型ロボット e-puck へ実装し、その精度を検証した。

２．方法

e-puck に図１のような軌道で走行するプログラムを組み込み、平らな机の上に固定した方眼紙上を走行させた。一定の走行距離毎に変化するロボットの中心位置をｘ、ｙ座標で記録し、Odometry によって得られた推定値と実際に測った実測値との比較を行った。

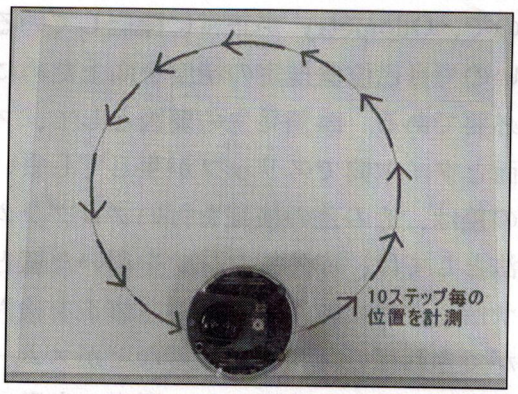

図１．e-puck が描く軌道

3．結果

得られたデータから、推定値と実測値が描く軌道を以下の図２に散布図としてまとめた。数値の単位は１ｍｍ、横軸はｘ、縦軸はｙである。

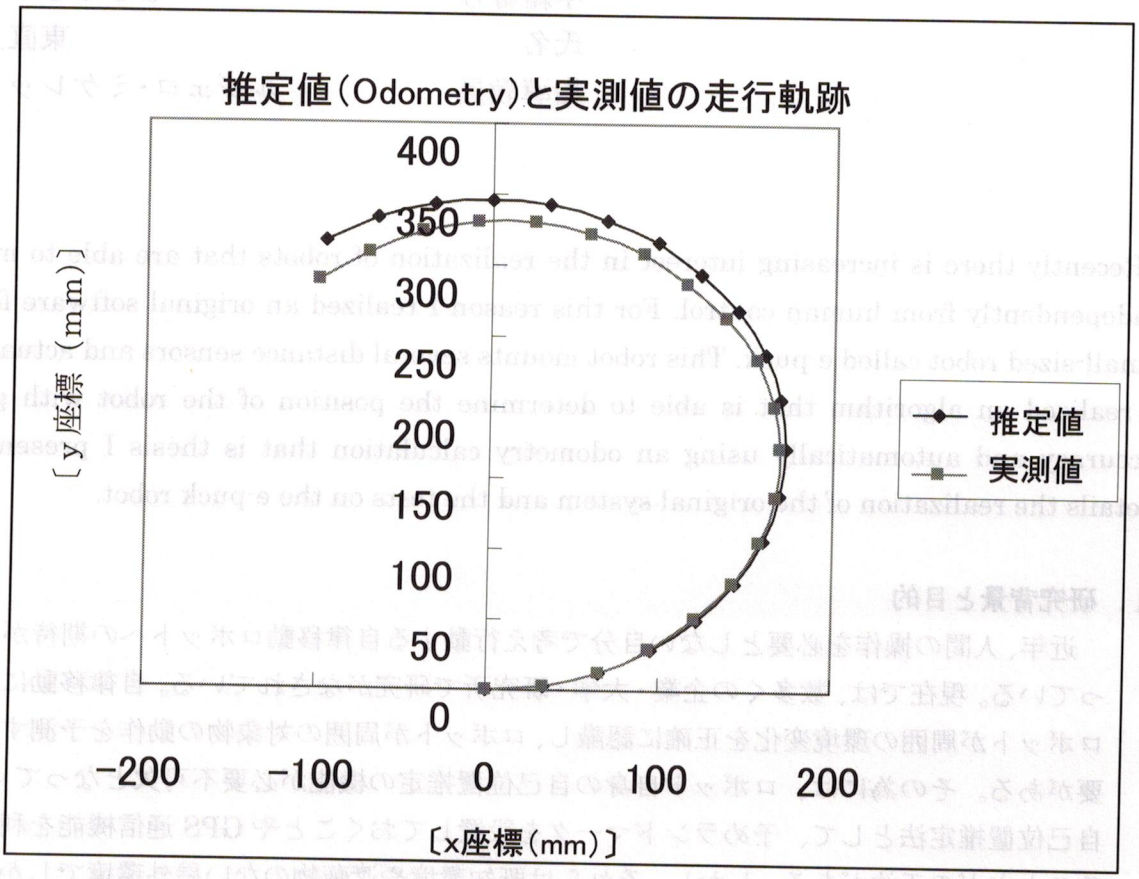

図２．推定値と実測値の軌跡

4．結論

Odometry で得られたデータにより e-puck の走行軌跡と類似した軌跡を得ることができたので、Odometry が正常に機能していることがわかる。しかし、誤差としてはまだまだ大きいので自己位置推定の精度を向上ためには、他のセンサを用いて誤差を修正するシステムが必要である。誤差発生の要因として、ステッピングモーターの制御能力に差があることや床面とタイヤ間でスリップが生じてしまい推定値がずれてしまうことが考えられる。制御能力の差は、その差の検証を行いプログラムで補正し改良を行う。スリップによる誤差修正の方法としては、赤外線センサを用いた壁との距離、カメラ機能を用いて得た画像、などのセンサ情報との比較により、誤差修正を検討していきたい。自己位置推定の精度を向上することができれば、障害物の回避などがスムーズに行え、また、応用として遠隔操作による環境地図作成などにも適用でき、幅広い応用の可能性がある。

＜卒業論文＞

「e - puck を用いた自己位置推定システムに関する研究」

横浜市立大学　国際総合科学部　国際総合化学科　基盤科学コース

ルジェロ・ミケレット　研究室

東　直人

目次

第1章　序論

1.1 はじめに

近年、人間の操作を必要としない自分で考え行動する自律移動ロボットへの期待が高まっている。現在では、数多くの企業・大学・研究所で研究がなされている。自律移動には、ロボットが周囲の環境変化を正確に認識し、ロボットが周囲の対象物の動作を予測する必要がある。その為には、ロボット自身の自己位置推定の機能が必要不可欠となってくる。現在、2輪駆動台車を用いる移動機構のロボットにおいては、移動量を車輪の回転数から求める手法が多く用いられている。しかし、これは車輪の滑りなどによってロボット自身が考える位置と実際の位置の間に誤差が生じてしまうという問題点がある。現在では、それを解決するため、他の機能やセンサを用いて誤差を修正し、より精度の高い自己位置推定を行う方法が検討されている。また、環境側に取り付けられたセンサの情報を用いた予測手法の研究も多く行われている。しかし、自律移動ロボットが実用化する上では、ロボットに搭載されたセンサのみを用いて自己位置推定を行うことが重要課題であるが、明確な手法はまだ確立されていない。よって、本研究ではロボットに搭載されたセンサ情報のみを用いて，様々な自律移動ロボットで適用可能である自律移動システムの構築を目指し研究を行う。

1.2 研究背景と目的

移動ロボットの自律化には、障害物との距離、スタート地点からどれだけ移動したかを記録し、次の行動をロボットが判断し行動することが必要である。そのため、ロボットが自らの位置を把握する自己位置推定が不可欠である。ゆえに、自律移動ロボットにとって自己位置推定は重要なシステムである。

前述したように自律移動ロボットにおける自己位置推定は、その対象となるロボットが搭載しているセンサの情報から位置を推定することが必要である。車両ロボットの場合、車輪の回転数からロボットの速度や位置を算出する方法が基礎的な方法として用いられており、これをオドメトリ（Odometry）法という。

以上より、本研究ではロボットの自律移動に欠かすことのできない機能である自己位置推定システムの開発を目的とする。屋内環境を対象に、小型ロボットの e-puck を使用し、自己位置推定法の中でも最も一般的手法で車両ロボットに利用されている Odometry を e-puck に実装し、推定値と実測値間の誤差を見積もり Odometry の精度の検証を行った。

第２章　ロボットの構成と運動

2.1　e－puck について

前章で述べた通り、今回の研究では e-puck を用いて実験を行った。e-puck はスイス連邦工科大学で開発された、研究開発及び基礎技術の取得・技術者育成の為のロボットである。
赤外線センサ、加速度センサ、カメラ機能、bluetooth など多様なセンサ、機能を有しており、人工知能の分野で研究試作機として使用されている。本研究では bluetooth 機能を用いて通信を行うことができる小型ロボット e-puck を用いて研究を行う。
e－puck の写真と、技術情報、操作例を以下の picture1,Table1,Table2 に挙げる。

picture1. e-puck の写真

技術情報　　**Table1**

特徴	技術情報
サイズ、重さ	直径 77mm,高さ 55mm、150g
プロセッサー	dsPIC 30F6014A
メモリー	RAM:8KB ; FLASH:144KB
モーター	ギア比 50:1 のリダクションギアのステッピングモーターを 2 個搭載
スピード	最高速度 15cm/s で進行可能
赤外線センサ	8 個の IR センサ
カメラ	480x640VGA カメラ
マイク	3 個搭載、音の位置を把握可能
加速度計	x,y,z 軸 3 方向の加速度計
LED ライト	リング状に赤色が 8 個、胴体に緑色が 1 個、前部に赤色が 1 個
スピーカー	WAV ファイルの音声データを再生可能
通信	bluetooth 通信が可能

＜e-puck の操作例＞ Table2

コマンド	説明
e_init_port()	ポートを標準の構成に初期化
e_set_led(unsigned int led_number, unsigned int value)	LED 番号を指定 (0-7) On または off の選択 (0=off 1=on それ以上=inverse)
e_led_clear()	全ての LED を消す
e_set_body_led(unsigned int value)	On または off の選択 (0=off 1=on それ以上=inverse)
e_set_front_led(unsigned int value)	On または off の選択 (0=off 1=on それ以上=inverse)
e_set_speed_left(int motor_speed)	左のモーターのスピード〔steps/s〕
e_set_speed_right(int motor_speed)	右のモーターのスピード〔steps/s〕
e_get_steps_left()	左のモーターのステップ数を読み取る
e_get_steps_right()	右のモーターのステップ数を読み取る
e_set_steps_left(int set_steps)	設定したステップ数に初期化する
e_set_steps_right(int set_steps)	設定したステップ数に初期化する
e_init_prox()	IR センサ機能の呼び出し
e_stop_prox()	IR センサの初期化
e_get_prox(unsigned int sensor_number)	IR センサの値を読み取る
e_get_ambient_light(unsigned int sensor_number)	環境光の値を読み取る
e_init_acc()	加速度センサ機能の呼び出し
e_get_acc(int *x, int *y, int *z)	加速度センサのアナログ値を読み取る
e_init_micro()	マイク機能の呼び出し
e_get_micro(int *m1, int *m2,int *m3)	マイクのアナログ値を読み取る
e_init_ad()	アナログ/デジタル変換機能の呼び出し
e_read_ad(unsigned int channel)	指定したチャンネルにアナログ/デジタル変換を行う
e_init_sound()	サウンド機能の呼び出し
e_play_sound(int sound_offset, int sound_length)	音の種類を選択 音の長さを設定
e_close_sound()	サウンド機能を終了する

2.2 ロボットの運動学

2.2.1 ロボットの移動機構

移動ロボットは大きく分ければ、1次元移動（有軌道、配管内などの移動）、2次元移動（地上、壁面、水面などの移動）、3次元移動（空中、水中移動）がある。移動ロボットとして多く実用化され、研究が行われているロボットは2次元移動ロボットであり、e-puckは2次元移動に分類される。

2.2.2 車輪式移動ロボットの運動学

車輪式運動ロボットは、自動車と違って慣性が問題となるほど早くは動かないので、加速度を考えなくてもよい。また、駆動車輪は指令の速度に達するまでほとんどの時間がかからないので、駆動速度は瞬時に変化させることができるとしてもよい。この課程のもとに車輪式移動ロボットの運動を定式化する。

車輪式移動ロボットの運動学は、車輪の転がり（場合によってはすべりを伴う転がり）によって移動するものであり、車輪速度とロボットの並進・回転速度との関係式で表現される。以下に関係式とその模式図を示す。

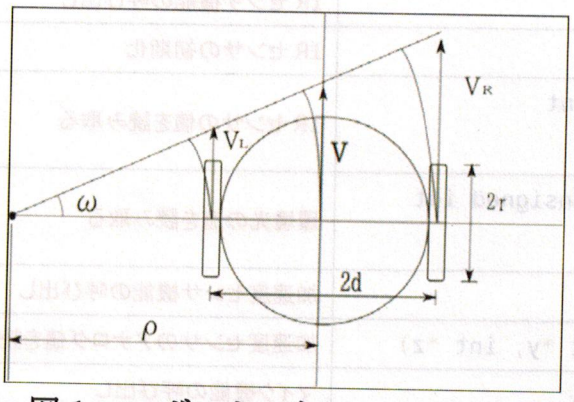

図1. ロボットの各パラメータ

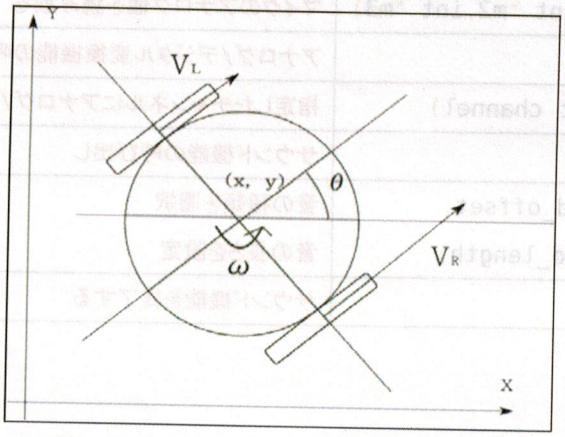

図2. 外部座標系で見た位置

図１のように各パラメータを定める。左右各々の車輪の回転角度をω_R、ω_Lとすると、車輪の接地点での速度は、車輪の半径をｒとして、

$$v_R = r\frac{d\omega_R}{dt}$$

$$v_L = r\frac{d\omega_L}{dt}$$

で得られる。ここで旋回中心回りの運動を考える。ロボットの旋回角速度をω、中心速度をｖ、旋回半径をρ、とすると、

$$v = \rho\omega$$

一方、中心から車輪までの距離をｄとすると、それぞれの車輪の旋回半径はｄだけ増減するため、

$$v_R = (\rho + d)\omega$$

$$v_L = (\rho - d)\omega$$

となる。この式を解いて、

$$\omega = \frac{(v_R - v_L)}{2d}$$

$$v = \frac{(v_R + v_L)}{2}$$

$$\rho = \frac{d(v_R + v_L)}{(v_R - v_L)}$$

が得られる。図２のように、ロボットを外部の座標系で見た場合は

$$\Delta x = v\cos\theta$$

$$\Delta y = v\sin\theta$$

$$\Delta\theta = \omega = \frac{v_R - v_L}{2d}$$

となり、ロボットの運動が両輪の回転角速度をもとに記述できる。

2.2.3 Odometory

上記で得られた式を積分することで、x、y、θを得ることができる。ロボットを制御するプログラムは一般に一定の時間間隔で車輪の回転情報を読み取り、モータを制御し、CPU は入力、演算、出力という順に処理を繰り返す。積分は微小時間を扱うことが前提であるが、積分を行う代わりに、この制御の１サイクル時間の運動を考える。これを制御周期と呼ぶ。

１回の制御周期の間に左右各々の車輪がv_R、v_L進んだ時、ロボット中心速度vは、

$$v = v_R - v_L$$

で表すことができる。
以上より、ロボットの初期の座標を、

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

とすると、ロボット中心がｐからｐ′に変化した座標は

$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}$$

と書ける。よって、ロボット中心座標ｐは下記の式で常に得ることができる。

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{v_R + v_L}{2} \cos(\theta + \frac{\theta}{2}) \\ \frac{v_R + v_L}{2} \sin(\theta + \frac{\theta}{2}) \\ \frac{v_R - v_L}{2d} \end{bmatrix}$$

第３章　e-puck への Odometry の実装

3.1　実験

3.1.1　準備

本実験で使用する e-puck は上記で述べたように 2 個のステッピングモーターを搭載している。今回使用するモーターの性能は、１回転あたり 20 ステップ・ギア比５０：１のリダクションギアである。よって、車輪が１回転すると 1000 ステップ進むことになる。

進んだ距離をℓとする。ℓの算出方法は、車輪の直径を R とすると、下記の通りになる。

$$\ell = \frac{\text{ステップ数}}{1000} \times R$$

e-puck のタイヤの直径はノギスを用いて計測した所、41.4mm であった。上記の関係を用いて算出される推定値の精度を検証するため、直線走行における推定値と実測値の差を比較した。その結果、推定値と実測値の差は±0.1mm 以内であった。よって、ステッピングモーターの精度は信頼して良い精度と言える。

3.1.2　方法

表面が平らな机を用意し、その上に方眼紙を貼り付け机との滑りが生じないように固定した。e-puck は Table2 より、e_step_speed のコマンドの引数として右は 200、左は 100 に設定し、図３のように円軌道を描くように走行させた。e-puck が最初にいる地点を x、y 座標の原点とし、10 ステップ進む毎に、推定値として Odometry によって算出された値、実測値として e-puck の位置座標の値、計 20 点を記録した。なお、今回使用したプログラムは参考付録に載せる。

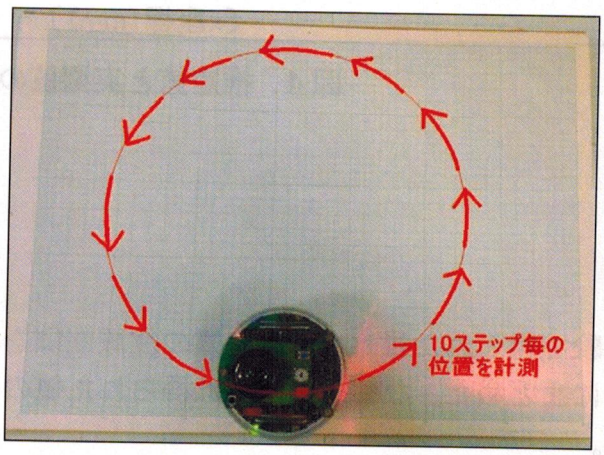

図３. e-puck が描く軌道

3.1.3 結果

推定値と実測値が描く軌道を以下の図４に散布図としてまとめた。数値の単位はｍｍ、横軸ｘ座標、縦軸ｙ座標である。最終到達地点での推定値と実測値の誤差は約30.4mm であった。

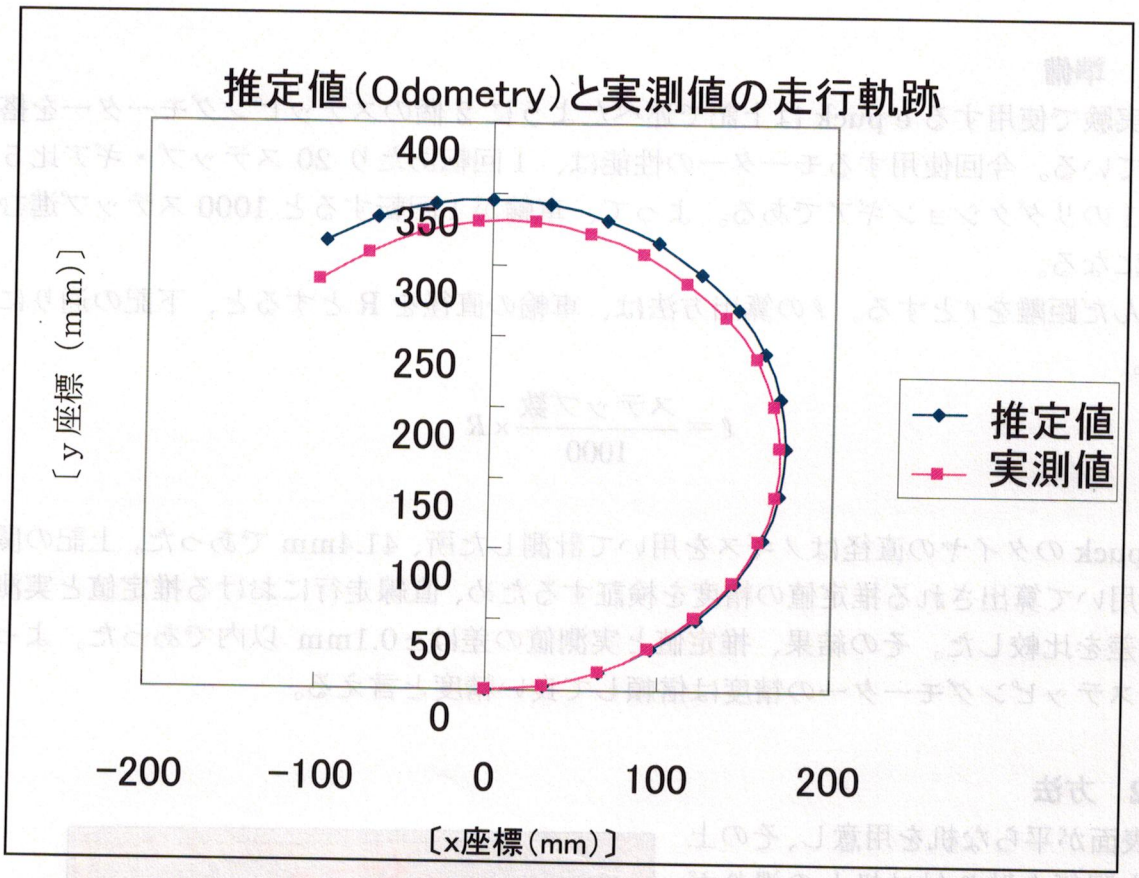

図４．推定値と実測値の軌跡

実験で得られた推定値、実測値の位置座標を次ページの Table3,Table4 にステップ数毎にまとめた。なお、推定値は得られた値の小数点第一位を四捨五入した値を表記した。

＜推定値＞ Table3

ステップ	x座標	y 座標
0	0	0
10	34	3
20	66	13
30	96	30
40	122	51
50	147	77
60	160	107
70	170	140
80	173	173
90	170	207
100	160	240
110	144	269
120	122	295
130	96	317
140	66	336
150	34	343
160	0	346
170	−34	345
180	−66	333
190	−96	317
200	−122	295

＜実測値＞ Table4

ステップ	x座標	y座標
0	0	0
10	33	3
20	65	12
30	94	29
40	121	52
50	142	76
60	158	106
70	167	138
80	169	172
90	165	202
100	155	235
110	137	264
120	114	288
130	88	309
140	58	322
150	25	330
160	−8	330
170	−40	324
180	−71	309
190	−100	289
200	−120	265

3.1.4 考察

得られた結果より、推定値と実測値が描く軌道は近似しておりOdometryの機能が正常に働いていると考えられる。一般的に言われる誤差の累積も確認することができた。しかし、直線とは違い、実際のロボットの動きは推定値の軌道よりも大きく内側に逸れていることがわかる。これには２つの原因が考えられ、ひとつは左右２つのステッピングモーター間の制御能力に差が生じていることが原因であると考えられる。内側に大きく逸れているので、設定した速度よりも左側のタイヤが遅くなっているか、右側のタイヤが速くなっている可能性がある。Odometry の制御をあげるためには、モーターの制御能力も十分に検証する必要がある。もう一つは曲がる際に内側の車輪がスリップしてしまっている可能性が挙げられる。これは内部センサの情報だけではスリップを感知するここができないので、このスリップの補正を行うシステムを検討しなくてはならない。

第4章　結論

本研究では、ロボットのよりスムーズな自律走行システムの開発に向けて、そのシステムの基盤となる自己位置推定方法、Odometry を e-puck に実装し、推定値と実測値間の誤差を見積もりその精度の検証を行った。Odometry で得られたデータにより e-puck の走行軌跡と類似した軌跡を得ることができたので、Odometry が正常に機能していることがわかる。直線で計測した推定値と実測値では非常に小さかった誤差が、曲がりを加えることで大きくずれてしまう原因として、ステッピングモーターの制御能力に差があること、曲がる際にタイヤがスリップしてしまっていることが挙げられた。今後の展望としてはOdometryにより得た推定値と、赤外線センサを用いて壁との距離やカメラ機能により得た画像などの情報との比較により、誤差修正システムを開発し自己位置推定の精度を向上させる。

また、e-puck は bluetooth 通信による遠隔操作も行えるので、遠隔操作による未知環境の環境地図作成（マッピング）などへの利用を応用研究として検討していきたい。

謝辞

まず、ルジェロ・ミケレット准教授に感謝の意を申し上げます。教授には、研究の知識とアイデアを提供して頂いただけでなく、様々な面でご指導いただきました。

そして、同研究室の及川虎太郎さん、岸野嵩久さんには大変お世話になりました。ご自身の活動が忙しいのにも関わらず、私の研究について親身になって相談に乗っていただきありがとうございます。

特に、及川虎太郎さんには様々なアイデアの提供、研究のアドバイスなど、言葉には表せないほどのご迷惑をおかけしました。

最後に、親身になってアドバイスしていただける准教授、先輩方がいる恵まれた環境の中で研究生活を送ることができたことに対し、感謝の意を申し上げて、本論文の結びとさせていただきます。

参考文献

［1］「詳説　ロボットの運動学」
オーム社　／　著　高野政晴

［2］岩波講座ロボット学１　「ロボット学創生」
岩波書店　／　著　井上博広、金井武雄、安西祐一郎、瀬名秀明

［3］「移動ロボットのための遡及的現在位置推定法－処理時間を要する外界センサデータの利用－」
日本ロボット学会誌　／　著　前山、大矢、油田

［4］「ロボットのための C 言語によるマイコン制御の考え方」
オーム社　／　著　城井田勝仁

［5］「ロボット制御工学入門」
コロナ社　／　著　美多勉

［6］「中部産業レポート Vol.6『次世代ロボット』報告書」
財団法人・中部産業地域活性化センター　／　著　篠田達彦

参考付録

```
/*****************************************************************************/
/**  オドメトリにより走行経路の推定値を得るプログラム                        **/
/**  実験日  2011/12/15                                作成者  東直人        **/
/*****************************************************************************/
#include "p30f6014A.h"
#include "e_epuck_ports.h"
#include "e_init_port.h"
#include "e_led.h"
#include "e_ad_conv.h"
#include "e_motors.h"
#include "e_prox.h"
#include "e_uart_char.h"

#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define M_PI 3.14159265358979           //円周率の定義
#define DELAY 50000                     //命令実行時間
#define WAIT 500000                     //命令の待機時間


//
//オドメトリ関数
//タイヤの回った角度から移動距離を求める
//
//        b : タイヤの幅      d_stepr:右車輪の微小変化     d_stepl:左車輪の微小変化
//      d_v : ロボット中心速さの微小変化    d_theta : ロボット回転角度の微小変化
//    vr,vl : 右、左の車輪が進んだ距離    cir : タイヤの円周
void odometry(double b,double d_stepr, double d_stepl, double cir, double *x, double *y, double
*theta)
{
        double vr,vl;
        double d_v,d_theta;

        double X,Y;
        double THETA;

        X = *x;
        Y = *y;
        THETA = *theta;

        //微小変化による各値の計算
        vr=d_stepr*cir/1000;
        vl=d_stepl*cir/1000;
        d_v=(vr+vl)/2;
        d_theta=(vr-vl)/b;


        //x座標、y座標の更新
        X= X+d_v*cos(THETA+d_theta/2);
```

```
        Y= Y+d_v*sin(THETA+d_theta/2);

        *x = X;
        *y = Y;

        THETA = THETA+d_theta;
        *theta = THETA;
}


int main() {



   long i,j;                                    //繰り返しの変数
   double x,y;                                  //x座標、y座標
   double wh_d=41.4;                            //車輪の直径[mm]
   double cir;                                  //車輪の円周[mm]
   double b=53;                                 //両輪の幅[mm]
   double stepr_now,stepr_past,d_stepr;         //右車輪；n番目のステップ数、n-1番目のステップ
                                                  数、n番目とn-1番目のステップ数の差
   double stepl_now,stepl_past,d_stepl;         //左車輪；n番目のステップ数、n-1番目のステップ
                                                  数、n番目とn-1番目のステップ数の差
   double theta;                                //ロボット中心の回転角度
   int pl,pr;                                   //左右の車輪の速さ
   char buffer[100];

   //各々の値を初期化
   x=0;
   y=0;
   theta=0;
   stepr_past=0;
   stepl_past=0;
   //円周を算出
   cir=wh_d*M_PI;

   //システムの初期化
        e_init_port();                          // ポートを標準の構成に初期化
        e_init_ad_scan(ALL_ADC);                // アナログ/デジタル変換機能を構成
        e_init_uart1();                         //  UARTを115200に初期化


   //フロントライトを付けて待機
   for(i=1;i<WAIT;i++){

        e_set_front_led(1);                     //フロントライトを点ける


            }

      //走行開始
      for(i=0;i<200;i++){

          pr=200;    pl=150;                   //車輪の速さを設定
```

```
        e_set_speed_right(pr);                  //右車輪を設定速度で回転開始
        e_set_speed_left(pl);                   //左車輪を設定速度で回転開始

        for(j=0;j<DELAY;j++)


        stepr_now=e_get_steps_right();          //右車輪；ｎ番目のステップ数
        stepl_now=e_get_steps_left();           //左車輪；ｎ番目のステップ数

        d_stepr = stepr_now - stepr_past;       //右車輪；ｎ番目とｎ－1番目のステップ差
        d_stepl=stepl_now-stepl_past;           //左車輪；ｎ番目とｎ－1番目のステップ差

        stepr_past=stepr_now;                   //右車輪；ｎ－1番目のステップ数更新
        stepl_past=stepl_now;                   //左車輪；ｎ－1番目のステップ数更新

        //オドメトリ関数で計算
        odometry(b, d_stepr, d_stepl, cir, &x, &y, &theta);

        //bluetoothを通じてPC側にデータを渡す
        sprintf(buffer,"%f,%f¥n¥r¥0",x,y);
            e_send_uart1_char(buffer, strlen(buffer));
        while(e_uart1_sending());
              }
}
```