「VRBOT D-1 を走らせよう」 解説資料
TechShare 株式会社
ロボット事業部

文責：柴田　拓哉

この資料では、VRBOT-D1 を使ったライントレースデモの詳細やプログラム解説をしています。

1. 使用するもの
   ・VRBOT D-1×1
   ・ライントレース用のコース
   （＊フォルダの「コース」をポスター印刷で A3 の 3×3 で印刷したものを使用します）

2. プログラム概要
   メインプログラムのそれぞれの関数の意味は以下のようになっています。

linedemo | AIStarter.h | Beep.cpp | Beep.h | E2PROM.cpp | E2PROM.h | HC-SR04.cpp | HC-SR04.h | IRModule.cpp

```
/*****************************************Copyright(c)*****************************************************
**                            Shenzhen Yuejiang Technology Co., LTD.
**
**                                 http://www.dobot.cc
**
**--------------File Info---------------------------------------------------------------------------------
** File name:            LinePatrol
** Latest modified Date: 2018-12-5
** Latest Version:       V1.0.0
** Descriptions:         Ai-Starter Demo
**********************************************************************************************************/
#include "AIStarter.h"

#define  IR_NUM    6              //赤外線の数の設定
#define  DBG_EN    0
#define  DBG_COLOR 0

/************* 初期設定 **************/
void setup()
{
    Serial.begin(115200);
    pinMode(13, OUTPUT);      //LED
    pinMode(36, INPUT);       //Bボタン
    AIStarter_SmartBotInit();
    Serial.println("SmartBotInit is completed");
    AIStarter_SmartBotSetLED(LED1,BLINK);
}

```

使用するピンの設定やシリアル通信時の転送速度の設定などを行っています

```
/*************赤外線センサーの状態を取得する **************/
void getCurrentIRState(int *irstate)
{
    *irstate = 0;
    for (int i = 0; i < IR_NUM; i++) {
        *irstate |= AIStarter_SmartBotGetIRModuleValue(i) << i;
    }
}

/************* 現在の体位を取得**************/
float getCurrentPos(const int irstate)
{
    const float coeff = 0.7;
    const int irPos[] = {-30, -18, -6, 6, 18, 30};//mm
    static float lastPos;
    float curPos;
    float readPos;
    int total = 0;
    int irOffCnt = 0;
    //calculate the car position offset
    for (int i = 0; i < IR_NUM; i++) {

        if (irstate & (1 << i)) {
            total += irPos[i];
            irOffCnt++;
        }
    }
    if (irOffCnt) {
        readPos = total / irOffCnt;
    }
    else {
        readPos = lastPos;
    }
    //calculate the current position
    curPos = (1 - coeff) * lastPos + coeff * readPos;
    lastPos = curPos;
    return curPos;
}
```

赤外線センサーで値の取得を行っています

赤外線センサーの値からVRBOT がラインに対してどの位置にいるか計算しています

```
67
68 /************** 设置小车速度 **************/
69 void setCarSpeed(const float curPos)
70
71 {
72     const int baseSpeed = 100; //rpm
73     //baseSpeed 70  kp 1  ki 0.06
74
75     //for speed 50
76     /*const float kp = 1;
77     const float ki = 0.06;
78     const float kd = 0.0;
79     const float errorsumLimit = 50;*/
80
81     //for speed 100
82     const float kp = 1.3;
83     const float ki = 0.3;
84     const float kd = 0.05;
85     const float errorsumLimit = 100;
86
87     float error = curPos;
88     static float lastError;
89     static float errorsum;
90     float errorChange;
91     int speedLeftWheel;
92     int speedRightWheel;
93     int speedOffset;
94
```

計算された位置を用いて、PID 制御をします。

```
94
95      //pid
96      errorsum += error;
97      if (errorsum > errorsumLimit) {
98          errorsum = errorsumLimit;
99      }
100     else if (errorsum < -errorsumLimit){
101         errorsum = -errorsumLimit;
102     }
103     errorChange = error - lastError;
104     speedOffset = kp * error + ki * errorsum + kd * errorChange;
105     lastError = error;
106
107     //calculate the wheel speed
108     speedLeftWheel  = baseSpeed + speedOffset;
109     speedRightWheel = baseSpeed - speedOffset;
110
111     //set tht wheel speed
112 #if DBG_EN
113     Serial.print("speedleftwheel = ");
114     Serial.print(speedLeftWheel);
115     Serial.print("    speedrightwheel = ");
116     Serial.println(speedRightWheel);
117 #endif
118     AIStarter_SmartBotSetMotor(MOTORL, speedLeftWheel);
119     AIStarter_SmartBotSetMotor(MOTORR, speedRightWheel);
120     Serial.println("go ahead");
121 }
122
```

PID 制御の続き

```
123 /************* 程序主循环 **************/
124 void loop()
125 {
126     if(!AIStarter_SmartBotGetKeyValue(37)) {
127         Serial.print("AIStarter_SmartBotGetKeyValue(37)=");
128         Serial.println(AIStarter_SmartBotGetKeyValue(37));
129         delay(5);
130         if(!AIStarter_SmartBotGetKeyValue(PINSW3)) {
131             digitalWrite(BEEP,HIGH);
132             delay(10);
133         }
134     } else {
135         if(AIStarter_SmartBotGetKeyValue(PINSW3)) {
136             digitalWrite(BEEP,LOW);
137         }
138     }
139     int irstate;
140     float curPos;
141     getCurrentIRState(&irstate);
142     curPos = getCurrentPos(irstate);
143     setCarSpeed(curPos);
144     delay(20);
145 }
```

プログラムのメイン
ループ文

このライントレースプログラムは主に PID 制御を使って制御をしています。
ここでは、PID 制御の解説をします。

(1) PID 制御とは

PID 制御は温度制御などで用いられることが多く、身の回りの家電製品にもよく使われています。

PID 制御では、自分が実際に設定する目標値(＊エアコンなどの設定温度)と制御した結果の実際の値との差(偏差)を利用します。この偏差の大きさや変化の速さに応じて制御を行い、偏差がなるべく 0 に近づけさせることが PID 制御の目標です。

PID にはそれぞれの制御方法があります。P は比例制御、I は積分制御、D は微分制御です。

(a) P：(比例制御) (図 1)

P の比例制御では、偏差の値の大きさによって制御対象(ここではモータ)に与える制御量を変化させます。つまり、偏差が大きいと制御対象に与えるエネルギーが大きくなり、偏差が小さいと制御対象に与えるエネルギーも小さくなります。これを、比例制御と言います。

しかし、上記のように偏差の値をそのまま制御量として使用することもありますが、偏差を何倍かし、制御する方法もあります。このときの倍率をゲイン(kp)と言います。ゲインを大きくすると早く目標値に近づくことができますが、ゲインが大きい過ぎてしまうと実際の値が安定しにくくなってしまいオーバーシュート(目標の値を超えてしまうこと)する恐れがあります。そのため、ゲインの調整が制御のカギとなります。

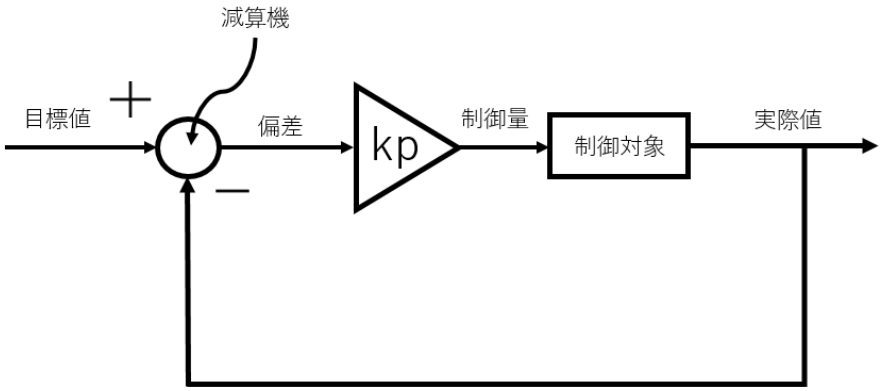

図 1　比例制御のブロック線図

(b) I：(積分制御)

比例制御だけでは、定常偏差というものができてしまいます。定常偏差とは、図 2 のように目標値に近づきはしますが、目標値の少し下の値で安定してしまい、目標値のそれ以上近づくことがないときの偏差のことです。

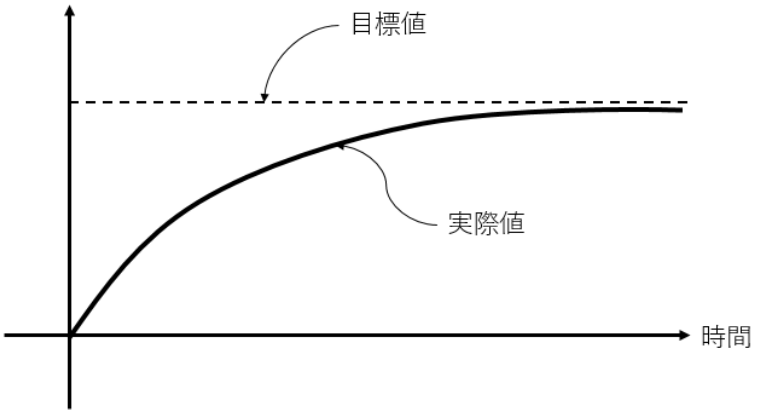

図 2　比例制御のみのグラフ

この定常偏差をなくし、目標値と実際値が一致させるためには積分制御を追加し、PI 制御を行います。

積分制御では、偏差を時間積分し制御量に加算します。たとえば、目標値を 3 V とし、一分ごとに誤差を足しこんだものを積分値とすると、以下の表 1 のように一分前の制御値に偏差を足したものが積分値となり、偏差と積分値を足した値が制御値となります。また、ここでの 0.2V を限界制御量とします。

表 1　積分制御の例

経過時間	実際値	偏差	積分値	制御値
0 分	2.8V	0.2V	0V	0.2V
1 分	2.8V	0.2V	0.2V	0.4V
2 分	2.9V	0.1V	0.5V	0.6V
3 分	3.1V	-0.1V	0.5V	0.4V
4 分	3.1V	-0.1V	0.3V	0.2V
5 分	3.0V	0V	0V	0.2V
6 分	3.0V	0V	0V	0.2V

積分制御も比例制御と同様にゲインをかけて積分制御の効果を調整します。全体のブロック線図を以下の図 3 に示します。

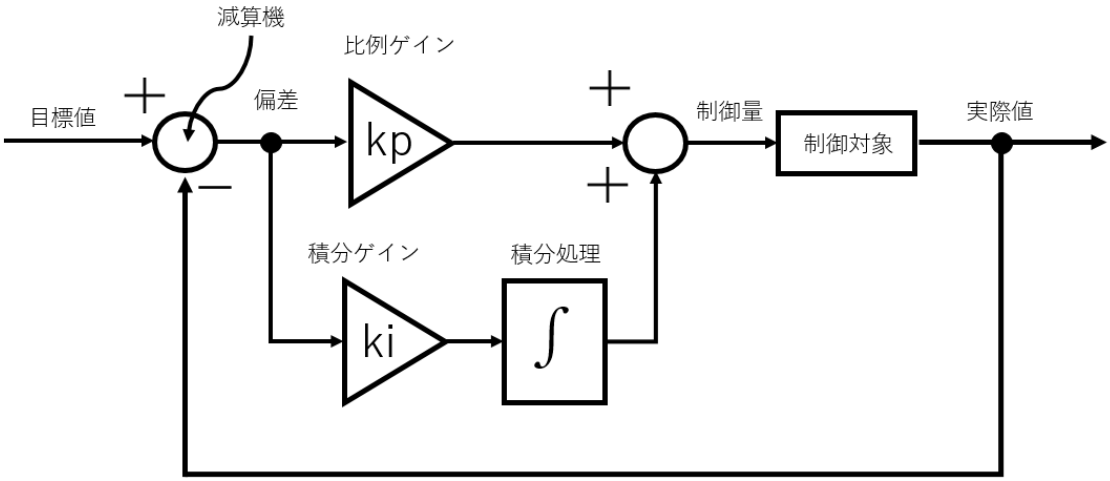

図 3　PI 制御のブロック線図

(c) D：(微分制御)

比例ゲインや積分ゲインを大きくし過ぎてしまった場合、オーバーシュートが起きやすくなってしまいます。そこで、微分制御を加えることで少し前の偏差と現在の偏差を比較し、急激に変化をしていた場合は偏差を小さくするように作用します。つまり、現在の偏差から少し前の偏差を引いて時間で割ると傾きが求められ、その傾きが大きくなったときにそれを打ち消す方向に出力をコントロールすることを微分制御と言います。微分制御でも今までと同様、微分ゲイン(kd)を用いて微分制御の効果を調整します。このとき注意する点は、微分ゲインをあまりにも大きくしてしまうと、目標値に到達していないにもかかわらず、目標値から離れる方向に制御量を変化させてしまうことが起きるため微分ゲインの調整には注意が必要です。全体のブロック線図を図 4 に示します。

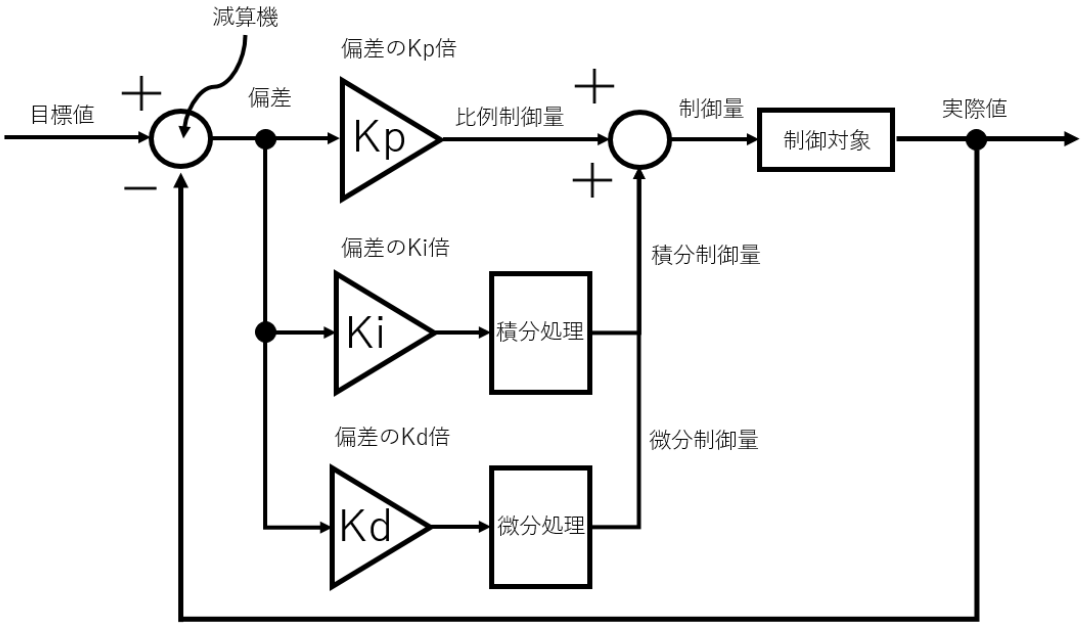

図 4　PID 制御のブロック線図

(2) PID 制御プログラム解説

図 5 にライントレースプログラムの一部があります。これまでに説明をしたように図 5 の PID のゲインの値を主に変化させることでライン上を上手く走れるかどうか決まってきます。以下を操作量の計算式の意味を表しています。

PID 制御の計算
青(比例)：Kp * error（ゲイン＊偏差）
赤(積分)：Ki * errorsum（ゲイン＊偏差を時間積分）
緑(微分)：Kd * errorchange（ゲイン＊傾き）

基本の速度(base Speed)を変化させると、それに対してゲインの値の変更を行う必要があるため注意してください。

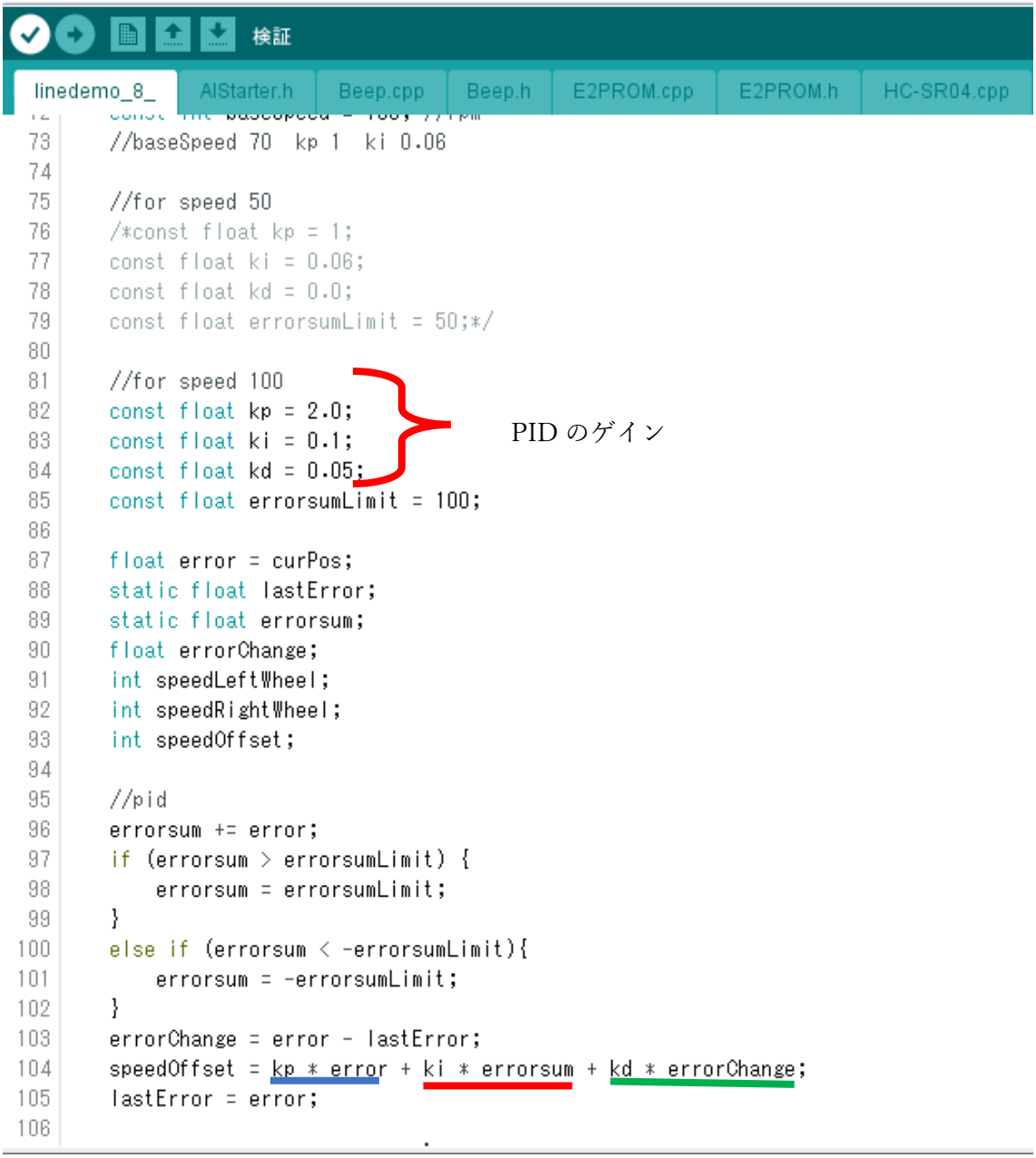

図 5　プログラム

TechShare 株式会社
ロボット事業部
初版：2019.02.08