実　験　報　告　書

実験題目　ライントレースロボットの製作

実験開始日	2020 年	10 月	8 日
実験終了日	2020 年	12 月	24 日
報告書提出日	2021 年	1 月	21 日

報告者

学籍番号　　氏名

TB18L036　　北一希

21/1/21 12 時 53 分

１　概要

白色の地面に対して引かれた黒いラインに沿って走行する自律走行型ライントレースロボットを製作する実験を行った．センサ，モータ，マイコンなどを利用しセンサが読み取ったデータを基に，マイコンに書き込んだ自作のプログラムによってモータを操作するロボットである．実験ではセンサ，モータの特性を理解し，マイコンに書き込むライントレースするためのプログラムを製作した．製作したプログラムのアルゴリズム，実際に競技会でロボットを走行させた結果と考察を記す．

２　目的

マイコン，センサおよびDCモータを用いることで，与えられたラインに沿って移動する自律制御型ライントレースロボットを製作することが目的である．

コースのラインをどのように読み取るのか，ロボットのタイヤをどのように操作するのかを思考し，競技会のコースを完走，走破時間を短くすることを目標とする．

３　原理

本実験で使用したロボットの主な構成要素はラインファインダ(センサ)，DC モータ(動作機構)，Arduino(マイクロコンピュータ)である．センサがコースのラインを読み取り，アナログ電圧出力する．この電圧に基づいてマイコンがラインと車体の位置関係を把握する．センサから与えられた数値によってモータをどのように操作させるかを書き込まれたプログラムによって，マイコンがモータを電圧によって操作する．

センサはロボットの先頭下部に3つ取り付けられており，地面から3 mmほど浮いている．DCモータによって回転するタイヤは後尾下部の左右に 2 つ取り付けられており，先頭に取り付けられているローラーと合わせて3点でロボットを支えている．マイコンは後尾上部に取り付けられている．マイコンのピンからセンサやモータドライバへ配線しており，センサから出力された電圧を参照しマイコンがプログラムに従いDCモータへ電流を流しタイヤを操作する．

４　実験に使用する機材

４．１　センサ

本実験で使用したセンサは QTR-3A Reflectance Sensor Array である．3 つのセンサはボードの長軸に沿って 9.525mm の間隔でボードに取り付けられている．各センサは，個別でアナログ電圧を出力する．このセンサはラインセンサーとして設計されており，汎用の近接センサまたは反射率センサとして使用することができる．各フォトトランジスタはプルアップ抵抗に接続されて分圧器を形成し，反射 IR の関数として 0V と 5V の間をアナログ電圧出力する．出力電圧が小さいほど，反射が大きいことを示す[1]．白を読み取った時の電圧は 0.21V，黒を読み取った時の電圧は 4.52V であった．

仕様

・寸法：32 × 8 × 3 mm
・動作電圧：5.0 V
・供給電流：50 mA
・出力フォーマット：3 つのアナログ電圧
・出力電圧範囲：0 V から供給電圧
・最適な検出距離：3 mm
・推奨される最大検出距離：6 mm
・ヘッダーピンなしの重量：0.6 g

４．２　DC モータ

本実験で使用した DC モータは FA-130 である．モータは導線に流れる電流によって回転する．モータには磁石が固定されており磁界が発生している．この中で電流が流れることによりローレンツ力が発生しモータはローレンツ力によって回転する．モータの回転数は印加される電圧が大きいほど大きくなる．

４．３　モータドライバ

本実験で使用したモータドライバは DRV8835 Dual Motor Driver Carrier である．0 V～11 V で 2 つのブラシ付き DC モータの双方向制御に使用することができる小型のデュアル H ブリッジモータードライバ IC である．チャネルあたり最大約 1.2 A を連続的に供給でき，最大電流に耐えることができる．チャネルあたり最大 1.5A で数秒間，比較的低い電圧で動作する小型モータの理想的なドライバである[2].

仕様

・モータ供給電圧：0V～11 V
・ロジック電源電圧：2 V～7 V
・出力電流：モータあたり 1.2 A 連続(1.5 A ピーク)
・モータ出力を並列化して，2.4 A の連続(3 A ピーク）を単一のモータに供給することができる．

４．４　マイコン

本実験で使用したマイコンは Seeeduino V4.2(ATMega328P)である．ボードは基本的に FTDI チップのように機能する．マイクロ USB ケーブルを介してボードをプログラムすることができ DC ジャック入力を介してボードに電力を供給することができる。7～15V の範囲で電圧を供給することができる．システム供給電圧 3.3 V または 5 V を選択するスイッチがあり，これは電力を節約するためにシステムを 3.3V に設定する場合に用いる．プログラムは Arduino 言語によって製作することができる．

仕様

・ATmega328P マイクロコントローラー
・ArduinoUNO ブートローダー
・14 個のデジタル I / O ピン（6 個の PWM 出力）
・6 つのアナログ入力
・ISP ヘッダー
・ArduinoUNO-R3 シールド互換
・マイクロ USB プログラミングと電源
・3 つのオンボード Grove コネクタ
・3.3 / 5V システム操作電源スイッチ[3]

５　方法

５．１　一番初めに考えたアルゴリズム

黒いラインに対するロボットの位置, つまり 3 つセンサの読み取った色が白か黒かで場合分けを行い, タイヤの操作量を決定するオンオフ制御を採用した．ただしセンサが黒か白か判定する電圧のしきい値は 2.7 V とした．このアルゴリズムのフローチャートを図 1 に示す.

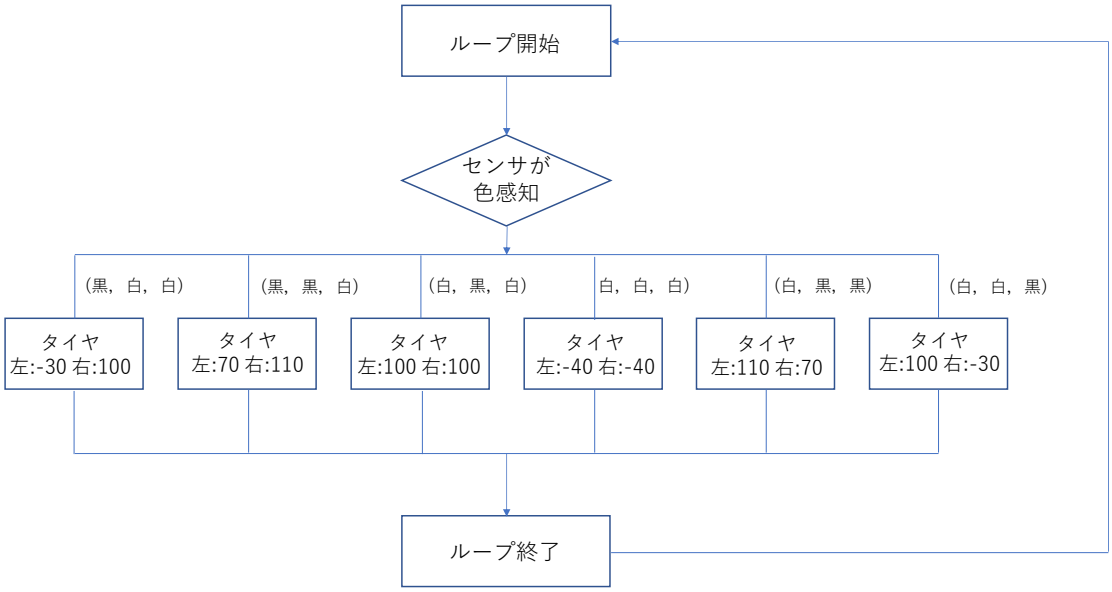


図 1 一番初めに製作したアルゴリズムのフローチャート

3 つのセンサのうち左側が黒色を読み取った場合，黒いラインはロボットの進行方向に対して左側に位置しているため，ロボットは左側へ曲がって進むべきである．同様に右側が黒色を読み取った場合，ロボットは右側へ進むべきである．3 つセンサのうち中央のみが黒色を読み取った場合，ロボットは黒いラインを順調にライントレースできているため，前方向に直進するべきである．ただしロボットが上記のようにカーブを行う場合，例えば 3 つのセンサが左から(黒，白，白)，(黒．黒，白)と読み取った場合では黒いラインからの逸れ具合に違いがある．前者の方が後者に比べ大きく逸れているため，より強い左カーブを行うべきである．右へカーブする場合においても同様のことがいえる．

またロボットがラインを見失いコースアウト，つまり 3 つのセンサが(白．白，白)を読み取ったときロボットは進行方向に対して逆へバックするようにした．これによりコースアウトした場合でもバック走行を行い，黒いライン上に復帰することが可能となる．

このアルゴリズムをもとに製作したプログラムを書き込んだロボットを走行させた結果，いくつかの課題が生まれた．

・無駄な動きが目立つ走行となった点

黒い直線ラインにおいて，ロボットが一度でも傾いてしまった場合，図 2 のようにジグザグとした走行を行ってしまい，滑らかな直進を行うことができなかった．これはラインに対する車体の逸れ具合に

適した軌道修正を細かく行えていない点，傾いた車体が(白，黒，白)を認識し直進したとしても傾いたまま行ってしまうため，ラインに平行な直進を行えない点が原因であると考えられる．この結果，無駄な動きが目立ち走破時間が伸びてしまう結果となってしまった．加えて直線のラインをこのように振動しながら走行してしまうため，直角コーナに対して車体が傾いた状態で進入してしまい，ラインの続く方向へ曲がることができずコースアウトしてしまうことがあった．

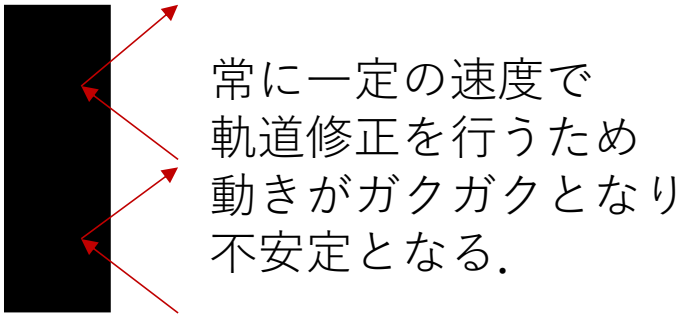
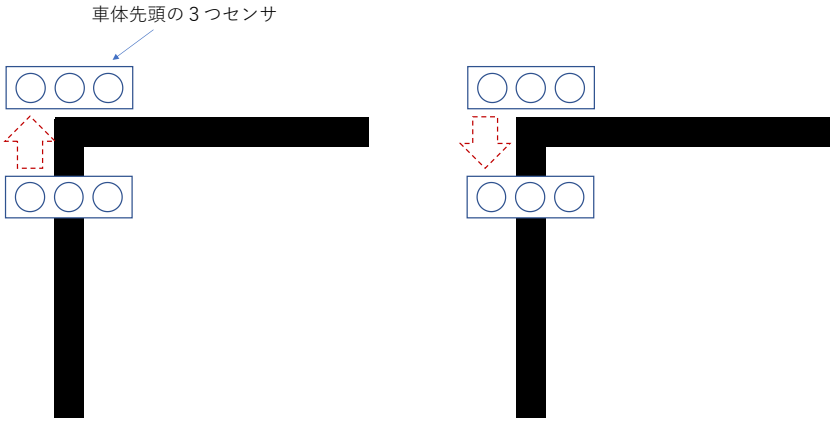


図 2　無駄の多い走行

・直角コーナにおいてバックと直進を繰り返しその場で振動してしまった点

直角コーナに対して強い勢いを持ち進入した場合，図 3(a)のようにコースアウトをした．センサが(白，白，白)と読み取った結果，バック走行を行うがコーナへ進入するときに通った黒いラインへ図 3(b)のように復帰し，その後再び図 3(a)のように直進しコースアウトしてしまった．この結果，車体はその場で振動し，ラインの続く方向へ走行できず走破することができなかった．またこの現象が発生した練習コース上の箇所を図 4 に示す．



(a)　　(b)

図 3　直角コーナにおけるロボットの動作

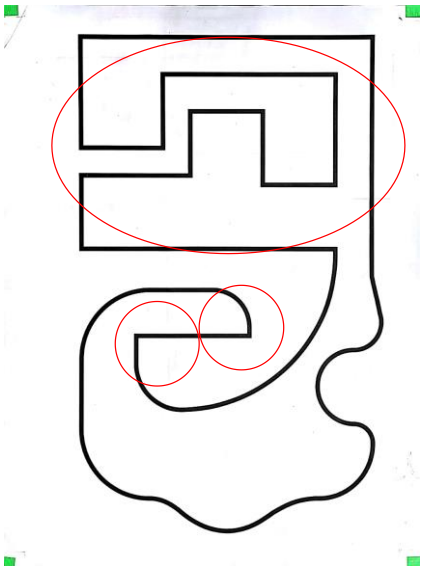


図 4　練習コースの直角コーナ(赤線で囲まれた部分)

５．２　最終的な完成版のアルゴリズム

最終的に製作したアルゴリズムでは，3 つのセンサが読み取った色によって場合分けし操作量を決定するオンオフ制御ではなく，センサの読み取った電圧の値によって代入する操作量を変化させる制御方法，PD 制御を採用した．ただしこれは直線やカーブにのみ対応することが可能で，直角コーナに対しては対応できなかった．そのため直角コーナに対しては引き続きオンオフ制御を採用した．このプログラムは付録に記す.

５．２．１　PD 制御

PD 制御とはセンサの読み取った電圧によってモータの操作量を変化させる制御であり以下の式で表される[4].

$$K_P \times \{V_1 - V_{target}\} + K_D \left\{\frac{d(V_2 - V_1)}{dt}\right\} \tag{1}$$

K_P，K_D:ロボットによって決定される定数 V_1:現在読み取っている電圧 V_2:前回読み取った電圧 V_{target}:センサが読み取るべき目標電圧，本実験では 2 V とした

この式を元々モータに定数の操作量を与えている状態に加える．第 1 項は P 制御，第 2 項は D 制御を表している．PD 制御は 3 つのセンサのうち左側と右側のものを用いる．左側のセンサが読み取った電圧によって左側のタイヤの操作量が，右側のセンサが読み取った電圧によって右側のタイヤの操作量が決定する．

P 制御は目標電圧と現在読み取った電圧との差に比例した操作量を求める[4]．オンオフ制御と比べ，

ラインからのロボットの逸れ具合，つまりセンサが読み取った電圧によって適切な操作量を決定するため図 5 のように無駄の少ない滑らかな走行を可能とする．これにより走行時間が短くなり，直角コーナに対してロボットがまっすぐ入射することができる．

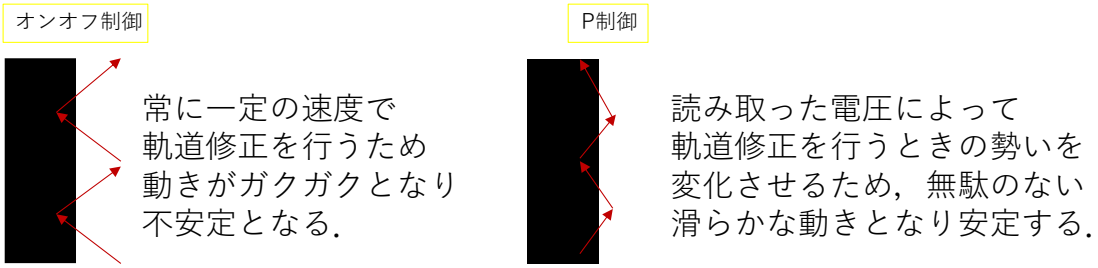


図 5　P 制御の利点

D 制御は前回読み取った電圧と現在読み取っている電圧の変化の割合から操作量を決定する[4]．この値は直線からカーブへ進入する瞬間に大きくなる値であり，急なカーブに対してカーブの初めに操作量が加わるため，これに対応することができる．P 制御のみでは過去に読み取った電圧を利用しないため急に形状が変わるコースに対応することが難しいため，この D 制御を採用した．しかし練習コースの直角コーナ(図 4)に対しては D 制御を用いても曲がり切れずにコースアウトしてしまった．

５．２．１　PD 制御に使用する定数の決定

PD 制御の式(1)における定数K_P，K_Dはロボットの重さや構成によって決定されるパラメータである．この定数を特定する実験の方法に限界感度法を行った．この実験によって正確な値の特定は難しいが簡単な方法で理論値に近い値を特定することができ，その後は実際にロボットを走行させることでチューニングを行う．この実験方法を以下に示す．

１　P 制御のみを用いて走行させる．

２　比例ゲインK_Pを増加させていく．ロボットは軌道修正をする勢いが増していく中で直線において初めロボットは振動を行い，その後徐々に直進を行うようになる．この値を増加させていくうちに直進を行わず常に振動する持続振動を行うようになる，このときのK_Pを記録する．

３　記録したK_Pの値と持続振動の周期から PD 制御を行う時のK_PとK_Dを求めることができる[5].

この実験の結果得られた持続振動の様子を図 9 に示す．このときK_Pは 63，周期は 0.54 (s)であった．この数値と表 1 の計算表によって実際の制御に用いる定数の値が決定する．ただし持続振動するときのK_PをK_U，周期をP_Uとする．

この結果，$K_P = 37.8$，$K_D = 2.55$となった．競技会ではこのままの値を使用した．

表 1　限界感度法の計算表

制御の種類	Kp	Ti	Td	Ki	Kd
P	0.50Ku				
PI	0.45Ku	0.83Pu		Kp/Ti	Kp*Td
PID	0.60Ku	0.50Pu	0.125Pu		

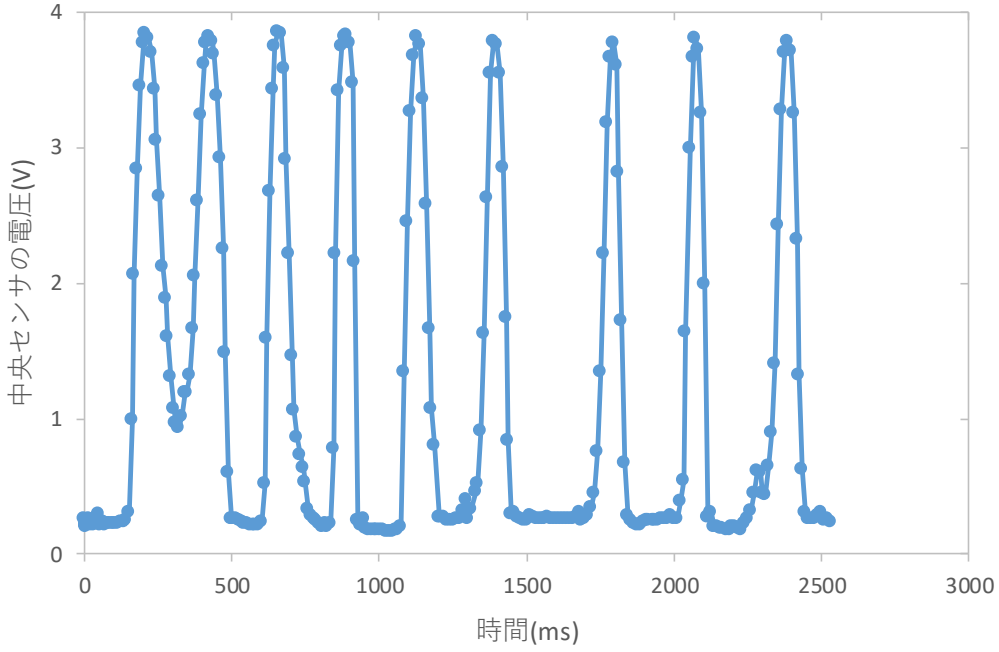


図 5　限界感度法における持続振動

５．２．３　直角コーナに対するオンオフ制御

ロボットは直角コーナへ入射するとき，速さが大きすぎる場合に曲がり切らずにまっすぐそのままコースアウトしてしまう．コースアウト後の操作として，真後ろへバック走行するように一番初めのアルゴリズムで対応していたが，最終的なアルゴリズムでは現在ロボットがどちらへカーブするべきかを記憶することでコースアウト直後に適切な方向へ走行することを可能とした．

プログラム中に変数 flag を用いる．この変数に代入されている数値(0 or 1)によってコースアウト，つまりセンサが(白，白，白)を読み取った時，右か左か適切な方向にロボットが回転するようにした．まずセンサが(黒，白，白)，(黒，黒、白)を読み取った場合，図 6 のように黒いラインがロボットの左側に位置しており，ロボットは PD 制御により左へカーブしているため．現在左へカーブするべきという意味で flag に 0 を代入する．この変数は次に数値を代入されるまでは保存されている．同様にセン

サが(白，白，黒)，(白，黒、黒)を読み取った場合，右へカーブするべきという意味で flag に 1 を代入する．ただしこのとき，センサが黒か白か判定する電圧のしきい値は 2.7 V とした．記憶をしている状態でコースアウトした場合，コースの続いている方向へロボットが回転するため，初めに考えたアルゴリズムのようにその場で振動することがなくなった．

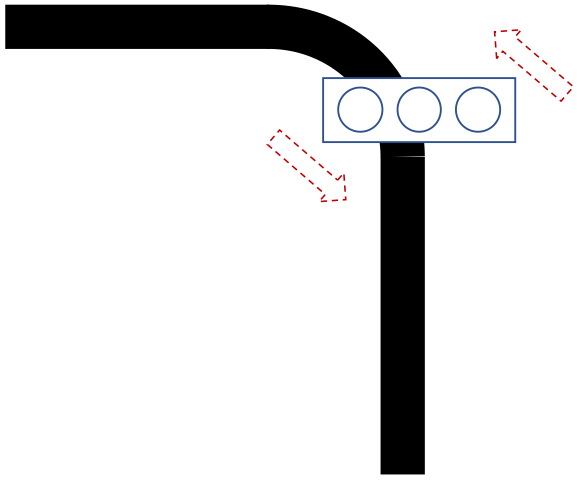


図 6　ラインとロボットの位置関係と回転するべき方向

ロボットが直角コーナに入射する，つまり図 7 のようにセンサが(黒，黒，白)，(白，黒，黒)と読み取っている場合と，コースアウトした場合にオンオフ制御を行う．ただしこのときセンサが色を判定する電圧のしきい値は 3.7 V とした．直角コーナへ入射する場合はっきりと 2 つのセンサが黒を踏むからである．

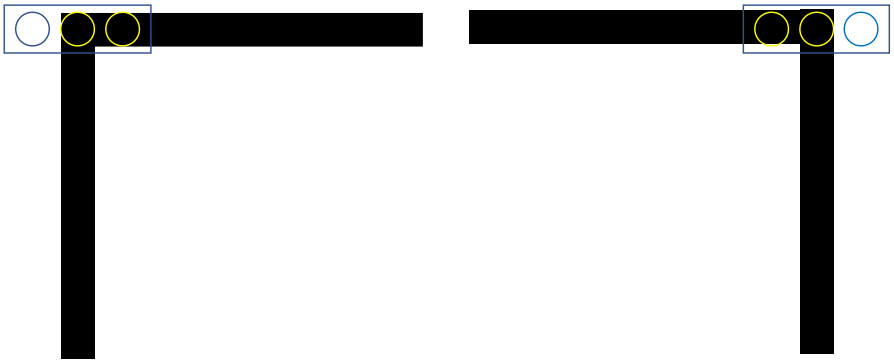


図 7　直角コーナとロボットの位置

オンオフ制御によって代入する操作量を表 2 に示す.

表 2　オンオフ制御の操作量

flagの値	センサの感知	左タイヤの操作量	右タイヤの操作量
	(黒，黒，白)	-60	60
	(白，黒，黒)	60	-60
0	(白，白，白)	-60	60
1	(白，白，白)	60	-60

直角コーナへ入射するとき，その場でコースの続く方向へロボットを回転するように制御させた．しかし回転する過程や勢いよく直角コーナへ入射した場合，センサがラインを見失い(白，白，白)と読み取るため flag の数値に従ってそのまま適切な方向へ回転を行うようにした．ただしその場でロボットが回転するように代入する操作量は 2 つのタイヤが同じ大きさで逆方向となるようにした．この操作量が大きいほど直角コーナの処理が早くなり完走時間が短くなるが，直角を曲がった後にロボットが傾いてしまうため，短い間隔で直角コーナが用意されている図 5 のコースにおいてロボットが傾いたまま直角コーナに入射してしまいセンサが直角コーナを認識することができないまま脱線してしまうことがあった．直角コーナを素早く処理すること，この脱線をしない安定感を保った操作量は 60～80 であった.

５．２．４　最終的な完成版アルゴリズムのフローチャート

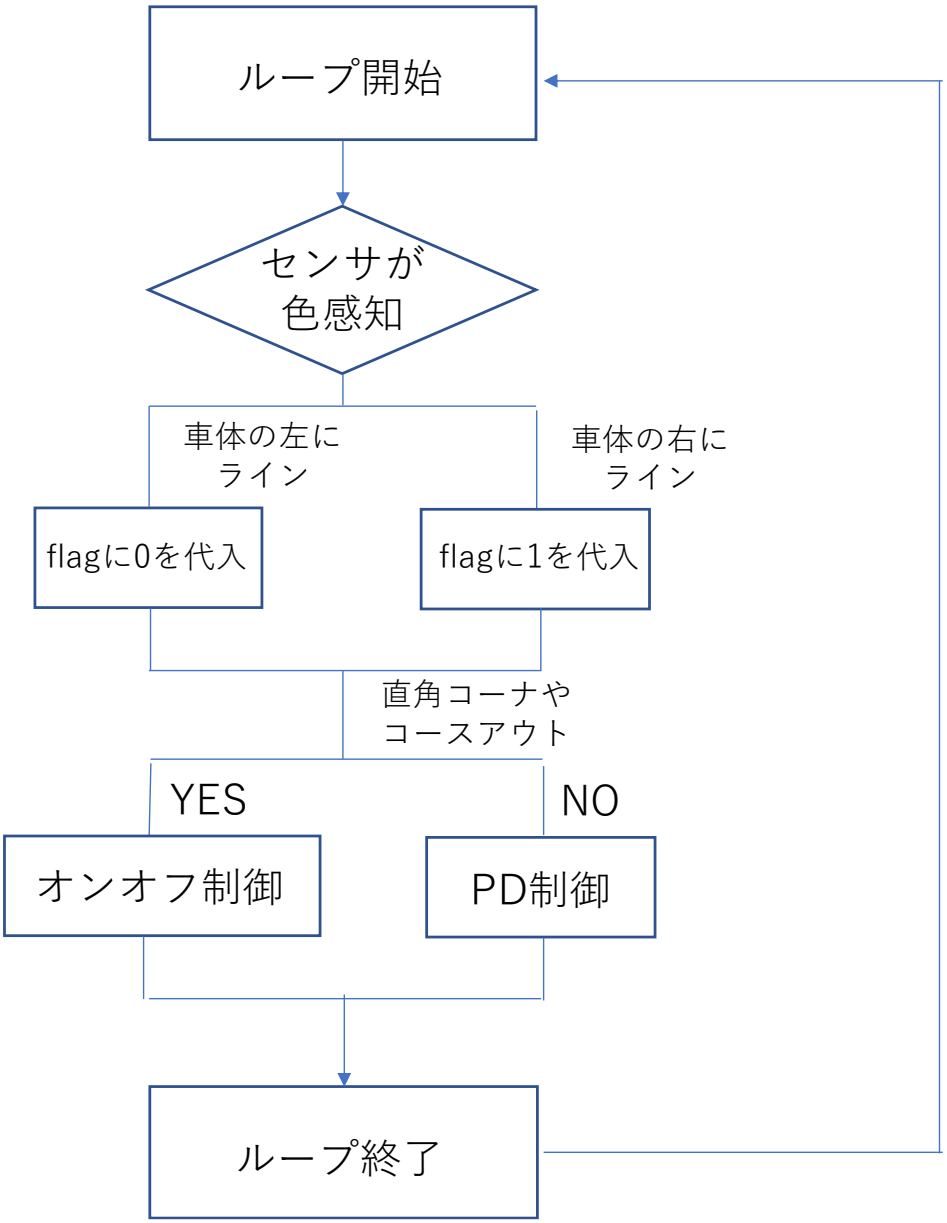


図８　最終的なアルゴリズムのフローチャート

6　実験結果と考察

競技会のコースを図 10 に示す.

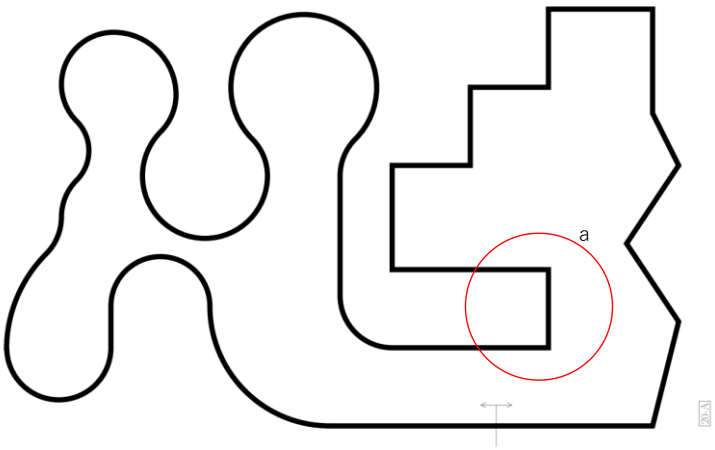


図 9　競技会のコース

持ち時間 5 分を 1 順として 5 順分の計 25 分間，最終的なアルゴリズムでロボットを走行させた．競技会の走行結果を表 3 に示す.

表 3　競技会の結果

	左回り	右回り
走行回数(回)	8	10
完走回数(回)	7	6
平均タイム(s)	46	43
ベストタイム(s)	43	40

ベストタイムは左回りで 43 秒,右回りで 40 秒であった. 走行回数 18 回のうち完走回数は 13 回で 5 回,完走することができず脱線した．1 順目に右回りで 2 回，2 順目に右回りで 1 回，3 順目に左右回りどちらも 1 回脱線をした．脱線した箇所は全て図 10 に示す a 地点のコの字の形をしたラインであった．1 から 3 順目ではコの字のラインで脱線する原因とその解決を目的として，4 順目以降は走破時間を短くすることを目的として走行を重ねた.

６．１　コの字での脱線の解決

コの字のラインでは短い間隔で 2 回直角に曲がらなければならないが，2 度目の直角でロボットが直角コーナに対して傾いた状態で入射したとき，センサが直角コーナを読み取ることができず，ラインが続いている方向とは逆の方向へロボットが回転してしまい脱線してしまうことがあった．この原因として直角コーナにおけるオンオフ制御での操作量，flag に値を代入するときの電圧のしきい値の 2 点に問題があると考えられた．これらの値を改善した結果 4 順目，5 順目では全ての走行で完走することができた．1 順目から 5 順目におけるこれらの値を表 4 に示す．

表 4　コの字での脱線を防ぐために調整した値

順目	1	2	3	4	5
電圧のしきい値(V)	2.7	2.7	2.4	2.4	2.4
オンオフ制御の操作量	±80	±80	±80	±60	±60

直角コーナでの操作量を 80 から 60 に小さくした結果，コの字での 1 回目の直角コーナで回転が緩やかとなり 2 回目の直角コーナへロボットが傾くことなく入射できるようになったこの結果，直角コーナでラインの続く正しい方向へロボットが回転することができるようになった．また flag に数値を代入するときセンサの読み取った電圧のしきい値を 2.7 V から 2.4 V へ小さくした．この結果，ロボットとラインのズレが以前より小さい場合でも flag に数値を代入するため適切な値を代入し更新し続けることで，間違った方向へのコース復帰をしなくなったと考えられる．

６．２　走行時間を短くする調整

4 順目から脱線をすることがなくなったため，4 順目，5 順目では走行時間を短くすることを目的として調整を行った．ロボットは直角コーナ以外では(定数操作量)+(PD 制御の操作量)で走行している．この定数操作量を調整することで走行時間の短縮を図った．140 から 180 の間で 10 ずつ値を変更した結果 160 のときベストタイムである 40 秒を記録することができた．180 のように大きすぎる値では直角コーナへ入射するときやカーブにおいて勢いが強すぎる結果，ロボットが大きくコースアウトすることが目立ち，コース復帰するためのオンオフ制御を行う時間が増えるため，無駄の大きい走行となってしまい走破時間が長くなってしまったのである．

7　結論

以上の過程からマイコン，センサおよびDCモータを用いることで，与えられたラインに沿って走行する自律制御型ライントレースロボットを製作することができた．直線やカーブに対してはセンサが読み取った電圧に対して操作量を変化させ制御を行うことで滑らかな走行を可能とするPD制御を採用し，PD 制御では対応の難しい直角コーナに対してはオンオフ制御を採用した．加えてラインを見失いコースアウトした場合の対処に対しても適切な方向へ車体を回転させることでラインを再発見するアルゴリズムを搭載させた．走行時間に関しては学年で優秀な成績を修めることができた．

実験中に最も難しく，そして達成できなかったと感じたことはPD制御についてK_P，K_Dの理想的な値を特定することである．限界感度法を行い大まかな値の特定には成功したと考えられるが，その後，実際に走行を重ねてチューニングすることができなかった．このチューニングに関しては目視で行い実験を繰り返す必要があるが，そのための時間を確保することができず，競技会中に調整することも難しかった．理想的な値を特定することができれば走行時間のベストタイムをより縮められたと考えられる．

参考文献

[1] QTR-3A Reflectance Sensor Array
https://www.pololu.com/product/2456 [閲覧日 2020/1/17]

[2] DRV8835 Dual Motor Driver Carrier
https://www.pololu.com/product/2135/ [閲覧日 2020/1/17]

[3] Seeeduino V4.2(ATMega328P)
https://www.seeedstudio.com/Seeeduino-V4-2-p-2517.html [閲覧日　2020/1/17]

[4] オンオフ制御の欠点を補う「PID制御」とは
https://monoist.atmarkit.co.jp/mn/articles/1005/21/news095.html [閲覧日　2020/1/17]

[5] 限界感度法
http://www.miyazaki-gijutsu.com/series/control332.html [閲覧日　2020/1/17]

付録　競技会で使用したプログラム（競技会でベストタイムを記録した時のもの）

```
#include <DRV8835MotorShield.h>
DRV8835MotorShield motors;

#define KP 37
#define KI 0
#define KD 2.8
#define target 2
#define CarSpeed_L 160
#define CarSpeed_R 160
#define C2 2.4
#define RA 3.6
#define spin 60

static signed long diff_L[2];
static signed long diff_R[2];
static float integral_L;
static float integral_R;
static int flag;
unsigned long time;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int sVal3 = analogRead(A3);
  int sVal4 = analogRead(A4);
  int sVal5 = analogRead(A5);
  float dt,pretime;

  float vR = sVal3 * (5.0 / 1023.0);
  float vC = sVal4 * (5.0 / 1023.0);
  float vL = sVal5 * (5.0 / 1023.0);
```

```
if(vL>C2&&vC>C2&&vR<C2)/*bbw*/
{
   flag=0;
}
else if(vL>C2&&vC<C2&&vR<C2)/*bww*/
{
   flag=0;
}
else if(vL<C2&&vC>C2&&vR>C2)/*wbb*/
{
   flag=1;
}
else if(vL<C2&&vC<C2&&vR>C2)/*wwb*/
{
   flag=1;
}

dt=(millis()-pretime)/1000;
pretime=millis();

time=millis();
Serial.print(time);
Serial.print("¥t");
Serial.print(vL);
Serial.print("¥t");
Serial.print(vC);
Serial.print("¥t");
Serial.print(vR);
Serial.print("¥n");

int LSPEED=L_PD(vR,dt);
int RSPEED=R_PD(vL,dt);

int ResultSpeed_L=CarSpeed_L+LSPEED;
int ResultSpeed_R=CarSpeed_R+RSPEED;
```

```
    if(vL<C2&&vC<C2&&vR<C2)
    {
      if(flag==0)
      {
        motors.setM1Speed(-spin);
        motors.setM2Speed(spin);
      }
      else if(flag==1)
      {
        motors.setM1Speed(spin);
        motors.setM2Speed(-spin);
      }
    }
    else if(vC>RA&&vR>RA)
      {
        motors.setM1Speed(spin);
        motors.setM2Speed(-spin);
      }
    else if(vL>RA&&vC>RA)
      {
        motors.setM1Speed(-spin);
        motors.setM2Speed(spin);
      }
    else{
        motors.setM1Speed(ResultSpeed_L);
        motors.setM2Speed(ResultSpeed_R);
    }
}

/*左のセンサによる PID*/
float L_PD(float vL,float dt){
    float p,i,d;

    diff_L[0]=diff_L[1];
    diff_L[1]=vL-target;
    integral_L+=(diff_L[1]+diff_L[0])/2*dt;
```

```
    p=KP*diff_L[1];
    i=KI*integral_L;
    d=KD*(diff_L[1]-diff_L[0])/dt;

    return    math_limit(p+i+d);
}

/*右のセンサによる PID*/
float R_PD(float vR,float dt){
    float p,i,d;

    diff_R[0]=diff_R[1];
    diff_R[1]=vR-target;
    integral_R+=(diff_R[1]+diff_R[0])/2*dt;

    p=KP*diff_R[1];
    i=KI*integral_R;
    d=KD*(diff_R[1]-diff_R[0])/dt;

    return math_limit(p+i+d);
}

float math_limit(float pid) {
    pid = constrain(pid, -35 , 150);

    return pid;
}
```