

情報メディア実験 B レポート  
ライントレーサロボットのフルスクラッチ実装

202111609  
仲村 和士

2024 年 2 月 9 日

# 目次

<b>第 1 章</b>	<b>はじめに</b>	4
1.1	目的 . . . . .	4
1.2	本レポートの構成 . . . . .	4
<b>第 2 章</b>	<b>電気回路解析の基本</b>	5
2.1	概要 . . . . .	5
2.2	電気回路の導入 . . . . .	5
2.2.1	導入 . . . . .	5
2.2.2	電流と電圧 . . . . .	6
2.2.3	オームの法則 . . . . .	7
2.2.4	理想電源 . . . . .	7
2.2.5	電力 . . . . .	8
2.3	直流回路の解析 . . . . .	8
2.3.1	電圧上昇と電圧降下 . . . . .	8
2.3.2	キルヒホッフの法則 . . . . .	9
2.3.3	閉路解析法 . . . . .	10
2.3.4	キルヒホッフの法則の問題点 . . . . .	11
2.3.5	分圧、分流 . . . . .	11
2.3.6	合成抵抗 . . . . .	12
2.3.7	電源の等価変換 . . . . .	13
2.3.8	重ね合わせの理 . . . . .	14
2.3.9	テブナンの定理 . . . . .	15
2.3.10	定理の利用例 . . . . .	15
2.3.11	テブナンの定理による解法 . . . . .	17
2.4	正弦波交流回路の解析 . . . . .	17
2.4.1	正弦波交流 . . . . .	17
2.4.2	交流回路上の素子 . . . . .	17
2.4.3	実効値 . . . . .	19
2.4.4	交流回路の例 . . . . .	19
2.4.5	複素記号法 . . . . .	20
<b>第 3 章</b>	<b>ハードウェア設計</b>	23

3.1	ハードウェア概観	23
3.2	回路図	23
3.3	部品表	24
3.4	ベース	25
3.5	光センサ部	25
3.6	光センサ拡張マイコン基板	25
3.7	メイン基板	26
3.8	電源基板	26
<b>第4章</b>	<b>ソフトウェア設計</b>	27
4.1	ソフトウェア概観	27
4.1.1	全体構成	27
4.2	光センサ値の検出	27
4.2.1	コース線の中心の推定	27
4.2.2	角度の推定	28
4.2.3	読み込みのボトルネック	29
4.3	速度の検出	30
4.3.1	エンコーダ	30
4.3.2	センサ値のサンプリング	30
4.3.3	生データから速さへの変換	31
4.3.4	非同期処理の実装	31
4.3.5	ノイズ対策	31
4.4	モータ制御	31
4.4.1	センサのPID制御	31
4.4.2	光センサにおけるPID調整前のパラメータ推定	32
4.4.3	アドホックな調整	33
4.4.4	Xbeeとの通信	34
4.5	ソースコード	34
4.5.1	光センサ関係(光センサ用ボード側)	34
4.5.2	光センサ関係(メインボード側)	35
4.5.3	エンコーダ関係	35
4.5.4	モータの制御関係	35
4.5.5	その他	36
<b>第5章</b>	<b>評価</b>	37
5.1	ループ速度	37
5.2	実測記録	37
5.3	感想	37
<b>参考文献</b>		39

<b>付録 A</b>	<b>設計の変遷</b>	40
A.1	零号機 . . . . .	40
A.2	初号機 . . . . .	40
A.3	ベースの設計 . . . . .	40
	A.3.1 Version 1 . . . . .	40
	A.3.2 Version 2 . . . . .	41
A.4	メイン基板 . . . . .	41
	A.4.1 Version 1 . . . . .	41
	A.4.2 Version 2 . . . . .	41
	A.4.3 Version 3 . . . . .	42
A.5	センサ値の統合 . . . . .	42
	A.5.1 Version 1 . . . . .	42
	A.5.2 Version 2 . . . . .	43
	A.5.3 Version 3 . . . . .	43
<b>付録 B</b>	<b>インシデント、問題解決集</b>	44
B.1	概要 . . . . .	44
B.2	レギュレータの周波数特性 . . . . .	44
B.3	PH コネクタ接触不良 . . . . .	44
B.4	回路設計のミス . . . . .	44
B.5	マイコンのバグ . . . . .	44
B.6	温度特性 . . . . .	45
B.7	何もしていないのに壊れた . . . . .	45
B.8	テスタの個体差 . . . . .	45
B.9	これらの組み合わせ . . . . .	45
<b>付録 C</b>	<b>ソースコード</b>	46

# 第1章

## はじめに

### 1.1 目的

本実験では、ライントレーサロボットの制作を通して、電気回路解析、制御工学に関する理論、および実際のソフトウェア、ハードウェアへの応用を学ぶことを目的とする。ライントレーサロボットは、周囲と明るさの異なるコース線（一般的には黒色または白色）を追従する非常に単純なロボットであり、最もシンプルな制御方法とキットを用いればロボット製作が初めての小学生でも容易に制作が可能なレベルである。しかし、フルスクラッチで設計し、高速で追従させようとすると必要な知識は広範にわたり、難易度は高くなる。本レポートでは、ロボット製作の過程での学習成果、および、実際のロボットの設計、製作方法、評価についてまとめる。

### 1.2 本レポートの構成

2章では、電気回路解析の基本についての学習成果として、直流回路および交流回路の解析手法をまとめる。2章の内容は一般的なものであるから、読む時間がない場合は飛ばして次の3章から読むことを推奨する。3章では、実際に走行体に使用したハードウェア設計について部分に分割して示す。4章では、ソフトウェア設計について、システム構成、制御手法および、部分ごとの実装について示す。これら設計に関する章ではこれから作ろうとする人に向けた解説という体で記述している。5章では、走行体の評価、振り返り、改善点、感想等について示す。

## 第 2 章

# 電気回路解析の基本

### 2.1 概要

本章では電気回路解析の基本的な方法について学習したことを解説する。学習には教科書 [1] を使用した。ロボットを作るときには、回路設計が必要不可欠である。正しい知識のもとに回路設計を行えば初步的なミスによる部品の破壊が減らせるほか、トラブルが発生したときの原因究明にも回路解析の知識が役に立つ。もちろん、ここで扱うすべての知識が今回のロボット製作に直ちに役に立つ訳では無い。たとえば、電気回路解析のメインテーマは直流回路ではなく交流回路であるが、今回のロボット製作で交流回路を利用している人はいないだろう。しかし、直流回路であっても DC-DC コンバーダのように交流に似た理論で動く部品は存在するし、発振器を使用した回路も決して珍しくない。コンデンサはどうだろうか。直流回路にコンデンサは登場しないが、実際に扱う回路は理想的ではないから過渡現象が存在することを忘れてはいけない。そして、電源基板のような部分には過渡現象を抑えるためにコンデンサが使われているが、それを解析するためには交流回路の知識があったほうが便利である。一通りの知識を抑えておくことで直接的には関係ないように思われる場面で役に立つことは非常に多い<sup>\*1</sup>。

本章では高校で学習するキルヒホッフの法則を紹介したあと、キルヒホッフの法則よりも楽に早く回路を解析するための手法を紹介し、キルヒホッフの法則からの脱却を図るというコンセプトで電気回路解析の基礎を解説する。

### 2.2 電気回路の導入

#### 2.2.1 導入

それでは早速始めよう。まず、電気回路にはどのような構成要素があるだろうか。非常にシンプルである。基本的には、電源、配線、抵抗という 3 つの要素しかないと考えて問題ない。これは直流回路に限らず、交流回路の世界ではコンデンサ、インダクタといった素子が登場するものの、驚くことにこれらの素子は一定の手続きを踏めば抵抗と同じように考えて解くことができる<sup>\*2</sup>。

<sup>\*1</sup> 学習量が膨大だったため、レポートに書ききれなかったことも多い。交流回路の電力、Quality factor、歪み波交流、過渡現象、磁気結合回路、4 端子回路、3 相交流回路、分布定数回路など。

<sup>\*2</sup> 注意点としては、電気回路の世界ではほとんどの場合「線形の回路」のみを扱う。線形な回路は線形な素子から構成される。少し難しく感じられるが、線形な素子であるかどうか大雑把に判定するな方法としてはオームの法則に従うかどうかを考えれば良い。豆電球のような非オーム抵抗は線形な素子ではない。その他にも、半導体素子であるトランジスタやダイオードなども線形な素子ではない。これらは電子回路の範囲で扱うものである。

次に、電気回路が解けるとはどのような状態であるだろうか。多くの場合、既知の素子を接続した回路上で任意の点の電流と、2点間の電圧が求められている状態を指す。よって、我々は「電流」と「電圧」について正しい理解をする必要がある。

## 2.2.2 電流と電圧

まずは電圧から考えよう。電磁気学を学んだ人に説明するなら、「単位電荷に換算した静電気力のポテンシャルエネルギーである電位の2点間の差」「電場を積分して求められる電位の2点間の差」などと言えば十分であるが、実は、電磁気学を学んでいない人に電圧の概念を正しく理解してもらうのは意外と難しい。たとえば、中学理科を説明しているトライさんのサイト [2] では、「電圧のイメージは、流れる電子1粒のいきおいのこと」「電流のイメージは、1秒間に流れる電子の数のこと」とされている。この文章から電圧と電流の違いがわかるだろうか？理解している人にとってはこの文章は大きく外してはいないことがわかる。しかし、勢いが強いと言われれば、電子（電荷）が高速で流れている状態をイメージしやすい。そして電荷のスピードというのはどうちらかというと電圧ではなく電流の定義である。やはりよくわからなくなってしまった。

そこで、状況を説明するのに昔から用いられているのが水流のモデルである。これも正しく意味を吟味できていないと大きな誤解を招く諸刃の剣であるが、この考え方では次のように置き換えることができる<sup>\*3</sup>。電源 $\leftrightarrow$ 水を上に汲み上げるポンプ、配線 $\leftrightarrow$ 水平な水路、抵抗 $\leftrightarrow$ 水路中につけた水車、として閉じた水路を形成する。このようにおけば、ポンプで水を持ち上げる高さ  $h$  に比例して単位時間あたりに水路中のある位置を通過する水の量は増加する。もし、 $h = 0$  であれば水が流れることはない。同じことが電気回路にも言える。電気回路では、高さにあたるのが電圧、単位時間に水路の断面を通過した水の量が電流、となる。注意点としては、「高さ」という量は常に2点間の差で決定されていて、1点で絶対的に決定されるわけではないということである。たとえばポンプで水を5m汲み上げたとき、その基準となっているのは持ち上げる前の位置である。逆に、ポンプで水を汲み上げたあの高さを基準の高さ 0 m とすれば、持ち上げる前の高さは-5 m である。おなじことが電圧にも言えて、電圧が 1.5 V の電池は負極を基準とすれば正極は 1.5 V であるが、正極を基準とすれば、負極は-1.5 V である<sup>\*4</sup>。電圧と聞くと水圧のように1点で決まるものであると誤解しやすい。そういう誤解をなくすために、電磁気学では電圧と同じ意味で「電位差」という言葉がよく使われることを覚えておくとよいだろう<sup>\*5</sup>。

次に電流について考えよう。水流モデルを考えれば、水流は  $(\text{水流}) = (\text{通過水量}) / (\text{経過時間})$  で表現できる。同様に電流を表現すれば、

$$I(\text{電流}) = Q(\text{電荷の通過量}) / t(\text{経過時間})$$

となる。これを拡張して微小時間でも同様の性質が成り立つと考えれば、

$$i(t) = \frac{dq(t)}{dt}$$

が成り立つ。

---

<sup>\*3</sup> 水流モデルには圧力に基づくモデルと高さの差に基づくモデルの二種があり、あまり明確に区別されずに用いられてきたことが文献 [3] で考察されている。

<sup>\*4</sup> 回路図中では電圧の基準点は GND(グラウンド) と書かれることが多い。

<sup>\*5</sup> 電位と電圧は似た使われ方をするので少々わかりにくいことがある。気持ち的には、明示的には書かれていないが、どこかに基準があることを前提としてある1点の電気的な高さを述べたいときには電位と表し、2点を明示してその電気的な高さの差について述べているときには電圧と書く。

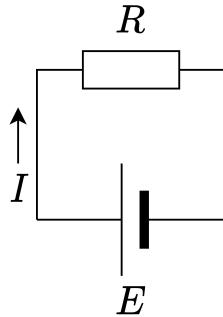


図 2.1

では抵抗はどのような装置であると考えられるか。抵抗はエネルギーを別の種類のエネルギーへと変換して取り出す装置である。水車は水の位置エネルギーを主に運動エネルギーへと変換する。同様に、電気回路中の抵抗は電気エネルギーを主に熱エネルギーへと変換する装置である。

### 2.2.3 オームの法則

図 2.1 ような単純な回路を考える。水流モデルと同様に、電気回路でも与えた電圧  $E$  に比例して電流  $I$  は増加する。このとき、以下の式が成り立つ。

$$E = RI$$

この係数  $R$  が抵抗である。同じ電圧ならば抵抗が大きいと電流は小さくなるため、抵抗は電流の流れにくさを表す。通常、 $R$  は時間によらず一定であるとみなして解析するが、実際の抵抗は温度変化による抵抗値の変動があるので注意する必要がある。 $R$  が一定であるとみなせるとき電圧と電流は比例し、このような関係は線形性とよばれる。電気回路の諸定理の中には線形性を満たすものにのみ適用できるものも多いから適用範囲に注意しなければならない。

また、抵抗の逆数  $G = 1/R$  を用いると、オームの法則は以下のように変形できる。

$$I = GE$$

この  $G$  をコンダクタンスとよび、電流の流れやすさを表す。

### 2.2.4 理想電源

#### 理想電圧源

回路の状態によらず、常に負極と正極の間の電圧が一定であり、一切の内部抵抗がない電源装置を理想電圧源と呼ぶ。

実際の電源装置は理想電源ではない。たとえば、生活の中で身近な直流電源装置として、電池があげられる。電池は利用しているとだんだんと消耗し、両端の電圧が下がっていく。また、瞬間に大きい電流が流れたときに両端の電圧が下がることがあり、やはり一定とは見なせない。そして、電池は内部抵抗が存在するため純粋に電源の機能だけとは見なせないのである。

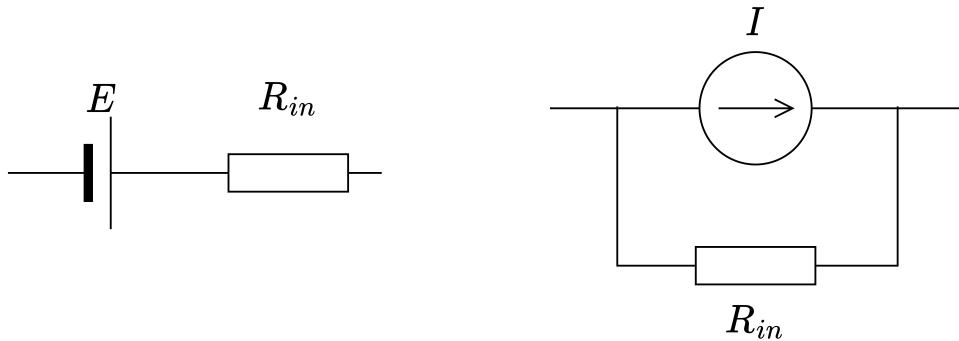


図 2.2 現実の電源の等価電圧源と等価電流源

### 理想電流源

回路の状態によらず、常に接続部分に一定の電流を流す電源装置を理想電流源と呼ぶ。

電圧と電流の因果律を考えれば、電圧があることが原因で電流が生じるから、電流源はより現実の電源装置からは離れた概念である。しかし、後述するように電源の等価変換を考えることにより回路解析では有用な場面がある。

### 現実の電源の等価電源

先に述べたように電池のような現実の電源は理想電源ではない。しかし、内部抵抗を考慮することで、より現実に近い電源モデルを考えることができる。図 2.2 のように、電圧源の場合は理想電圧源  $E$  と直列に抵抗  $R_{in}$  を挿入し、電流源の場合は理想電流源  $I$  と並列に抵抗  $R_{in}$  を挿入することで、内部抵抗を考慮した電源モデルとなる。

### 2.2.5 電力

電気が単位時間に行う仕事、すなわち、仕事率のことを電力とよぶ。ある抵抗  $R$  で消費される電力  $P$  は、抵抗にかかる電圧  $V$  と抵抗に流れる電流  $I$  を用いて、以下のように表される。

$$P = VI = RI^2 = V^2/R$$

電力の単位は W (ワット) であり、[W] = [J]/[s] である。

## 2.3 直流回路の解析

### 2.3.1 電圧上昇と電圧降下

オームの法則を用いることで、単純な回路については解析することができる。

ここで、回路解析のうえで重要な概念である「電圧上昇」と「電圧降下」の概念について説明する。図 2.3 のような回路を考える。ただし電圧の矢印は電位の低い方から高い方へ向けて書くことと約束する。この回路において、 $1'$  を基準とした各端子の電位を考える。電流の向きに沿って考えると、まず、 $1'$  から  $1$  へ移動する間の電源で  $E$  の電圧上昇があるから  $1$  の電位は  $E$  である。次に、 $1$  から  $2$  に移動する過程では抵抗  $R_1$  によ

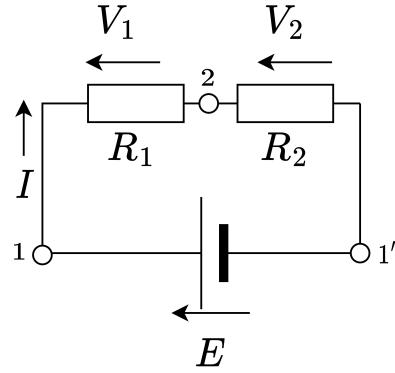


図 2.3

り、 $V_1 = R_1 I$  の電圧降下がある。そして、2 から 1' に戻る過程で抵抗  $R_2$  により、 $V_2 = R_2 I$  の電圧降下がある。これらから方程式を導くと、

$$\begin{aligned} E &= V_1 + V_2 \\ &= R_1 I + R_2 I \end{aligned} \quad (2.1)$$

となる。つまり、

$$(\text{電圧上昇の和}) = (\text{電圧降下の和})$$

あるいは、

$$(\text{電圧上昇の和}) - (\text{電圧降下の和}) = 0$$

が成り立つ。

次に、同じ意味であるが他の見方を示そう。図 2.3 を見ると、電源電圧  $E$  は電流の方向に沿って上昇している一方で、抵抗  $R_1, R_2$  にかかる電圧は電流と逆の向きに沿って上昇していることがわかる。このような状況は、抵抗が「逆向きに接続された電源」のように振る舞っていると解釈することができる。この逆向きの電圧を「逆起電力」と呼ぶことがある<sup>\*6</sup>。これを式にすれば、

$$(\text{電源による起電力の和}) = (\text{逆起電力の和})$$

となり、これは前に示した電圧上昇、電圧降下によるモデルと等価な式である。

### 2.3.2 キルヒ霍ッフの法則

オームの法則により、ループが 1 つしかない回路については解くことができる。より一般的に、ループが複数の回路でも適用できる法則がキルヒ霍ッフの法則である。

#### キルヒ霍ッフの電圧則 (KVL)

回路中にある閉ループをとる。どのような経路を通ってきてもループを 1 周して戻ってきたときの電位は最初と同じである。

---

<sup>\*6</sup> 抵抗による逆起電力という言葉は交流回路で登場するインダクタ（コイル）による逆起電力と本質的に別の仕組みであるため、混同を防ぐため使わない方がよいという意見もある。回路方程式を立てるうえでは両者とも電流の逆向きに沿って正の起電力があると見なせるので同じ手続きで問題ない。

つまり、より複雑な回路についても、任意にループを定めたとき、先に述べた式

$$(電圧上昇の和) - (電圧降下の和) = 0$$

が成り立つ。

一般にループ内に  $n$  個の電圧上昇があって  $i$  番目の電圧上昇が  $E_i$  と表され、 $m$  個の電圧降下があって  $j$  番目の電圧降下が  $V_j$  と表されるとき、以下の式が成り立つ。

$$\sum_{i=1}^n E_i - \sum_{j=1}^m V_j = 0$$

### キルヒ霍フの電流則 (KCL)

ある点で回路が分岐しているとき、その点に流入する電流の総和とその点から流出する電流の総和は等しい。つまり、分岐点に電荷が蓄積されることはない。

これを式に表そう。ある分岐点に  $n$  個の分岐があり、 $i$  番目の線に流れる電流を  $I_i$  とする。ただし、流入する方向を電流の正方向とする。このとき、次の式が成り立つ。

$$\sum_{i=1}^n I_i = 0$$

### 2.3.3 閉路解析法

キルヒ霍フの法則を使えば理論的にはどのような回路を解くこともできる。しかし、どの部分の値(電圧、電流)を仮定して、どのような立式をすればうまく解けるのかということは述べられていない。ここでは、統一的な手順で回路を解くことができる方法のひとつとして、閉路解析法を紹介する<sup>7</sup>。この方法は KVL を用いた方法であるから、電源がすべて電圧源である必要がある<sup>8</sup>。

図 2.4 のような回路を考えよう。この回路で各点に流れる電流はどのように表されるだろうか。

この回路で、まず、2つの閉路に流れる電流(閉路電流、またはループ電流とよぶ)を仮定し、その2つの閉路に成り立つ方程式を導くと以下のようになる。

$$\begin{cases} E - R_1 I_1 - R_2(I_1 - I_2) = 0 \\ -R_2(I_2 - I_1) - R_3 I_2 = 0 \end{cases}$$

これを解くことで、 $I_1 = \frac{R_2 + R_3}{R_1 R_2 + R_2 R_3 + R_3 R_1} E$ ,  $I_2 = \frac{R_2}{R_1 R_2 + R_2 R_3 + R_3 R_1}$ を得る。

一般的な回路において、仮定が必要な閉路電流の数  $l$  はグラフ理論の結果を用いることでわかる。電気回路を節と辺で構成されたグラフと同一視する<sup>9</sup>と、 $l$  は節の数を  $n$ 、辺の数を  $b$  とすれば、以下の式が成り立つ。

$$l = b - n + 1$$

<sup>7</sup> 節点解析法という KCL をベースとした手法もあるが、閉路解析法の方がシンプルで使いやすいことが多い。本稿では節点解析法は省略する。

<sup>8</sup> 後に紹介する電源の等価変換を用いれば電流源があっても解くことができる。

<sup>9</sup> 何を節と辺に取るかというのは色々なやり方があるようで、たとえば、素子を節に取るやり方や、配線の交わる点を節に取るやり方がある。個人的には後者の方がわかりやすいと思う。配線の交わる点とはある点から 3 つ以上の分岐があるものを指しているが、仮に 2 つしか分岐がない点(つまり配線の交点ではない点)を節にとっても、その分辺の数も増えるため結果は変わらない。

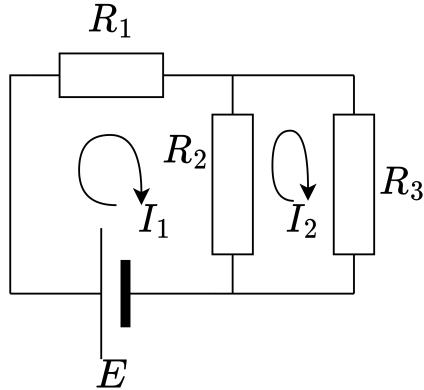


図 2.4

### 2.3.4 キルヒ霍ッフの法則の問題点

キルヒ霍ッフの法則は理論的には万能であり、十分な情報が与えられていればすべての回路を連立方程式に帰着させて解くことができる。

しかし、手計算で回路を解く場合、計算ミスをしやすいうえ、最善手とはならず、もっと楽に解く方法がある場合も多い。なぜなら、キルヒ霍ッフの法則では一般的に、仮定したすべての未知数を求めるまで解が定まらない。実際に回路を解きたいときは、ある部分のみを知りたい場合も多いから遠回りになってしまふのである。

ここからは、様々な解析手法を学び、キルヒ霍ッフの法則から卒業しよう。

### 2.3.5 分圧、分流

#### 分圧

キルヒ霍ッフの方法から直ちに求められる公式が分圧、分流の式である。いま、再び 2.5 に示す直列回路を考えよう。この回路についてキルヒ霍ッフの法則を用いて解析すれば、以下の式が成り立つことが直ちにわかる。これを用いれば単純な直列回路ではキルヒ霍ッフの法則を用いる必要がなくなるので時間短縮に役立つ。

$$V_1 = \frac{R_1}{R_1 + R_2} E$$

$$V_2 = \frac{R_2}{R_1 + R_2} E$$

#### 分流

次に、図 2.6 の並列回路を考えよう。この回路では  $E$  から全体に電流  $I$  が流れている、分岐点で  $R_1$  側に  $I_1$ 、 $R_2$  側に  $I_2$  が流れている。これらの電流の関係を考えると、以下の式が成り立つ。

$$I_1 = \frac{R_2}{R_1 + R_2} I$$

$$I_2 = \frac{R_1}{R_1 + R_2} I$$

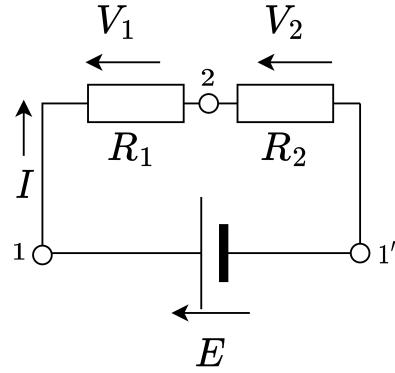


図 2.5

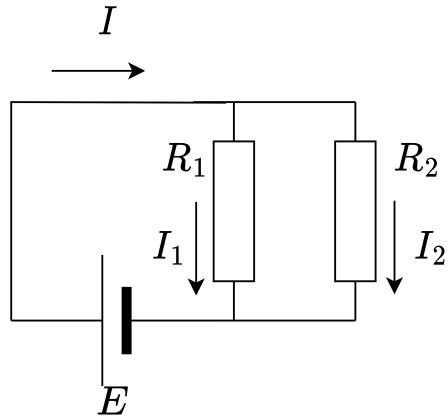


図 2.6

分圧の式と比べると、分子の抵抗が逆になっていることがわかる。これらの式は形が似ているので覚えにくいかかもしれない。そういうときには片方の抵抗を 0 や  $\infty$  であると考えてみると正しい式かどうか確認することができる。

### 2.3.6 合成抵抗

複数の抵抗が直列や並列に配置されており、個々の抵抗に流れる電流や電圧には興味がないが、全体で流れている電流を求めたいことがあるだろう。あるいは、分流を求めるために先に全体の電流を求めたいことがあるかもしれない。そういう際には、回路に接続された複数の抵抗をまとめて 1 つの合成抵抗とみなして解析する必要がある。

直列回路から始めよう。図 2.5 の回路において、2 つの直列抵抗をまとめてひとつの合成抵抗  $R$  とすると、オームの法則より、

$$E = RI$$

また、式 (2.1) より、

$$E = (R_1 + R_2)I$$

これらの 2 式を比較すると、

$$R = R_1 + R_2$$

である。直列に接続された合成抵抗は単純に和を取れば良い。

次に、並列回路について考えよう。図 2.6 において、次の 3 式が成り立つ。

$$\begin{aligned} E &= R_1 I_1 \\ E &= R_2 I_2 \\ I &= I_1 + I_2 \end{aligned}$$

これら 3 式から、 $I = (\frac{1}{R_1} + \frac{1}{R_2})E$  が導かれ、これを合成抵抗  $R$  を用いたオームの法則と比較すると、

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$$

よって、

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

が成り立つ。

ここで、並列回路の合成抵抗をコンダクタンス  $G=1/R$  を用いた式で表すと、以下のようになる。

$$G = G_1 + G_2$$

直列回路では抵抗を用いた式が簡単に表されるが、並列回路ではコンダクタンスを用いると簡単に計算できる場合がある。

### 2.3.7 電源の等価変換

2.2.4 節で電源には 2 種類あることを説明した。これらは互いに相容れないものであるわけではなく、相互に変換することができる。改めて電源の図 2.7 を示す。図の左の電圧源を右の電流源に変換するときには、以下のようにおけば良い。

$$\begin{aligned} I &= \frac{E}{R_1} \\ R_2 &= R_1 \end{aligned}$$

逆に、電流源を電圧源の形に変換するときには以下のようにおけばよい。

$$\begin{aligned} E &= IR_2 \\ R_1 &= R_2 \end{aligned}$$

これらの証明は 2.3.9 節で紹介するテブナンの定理を用いると簡単に与えられる。

この変換規則を見たとき、実際の回路には複数の抵抗があるが、どこまでを内部抵抗として扱えば良いのか疑問をもつ人がいるかもしれない。結論から言うとそれは自由であり、どこまでの抵抗を内部抵抗として扱っても問題ない。後に説明するが、この変換規則と合成抵抗の公式を繰り返し利用して回路を簡単な等価回路に置き換えて解くことができる。

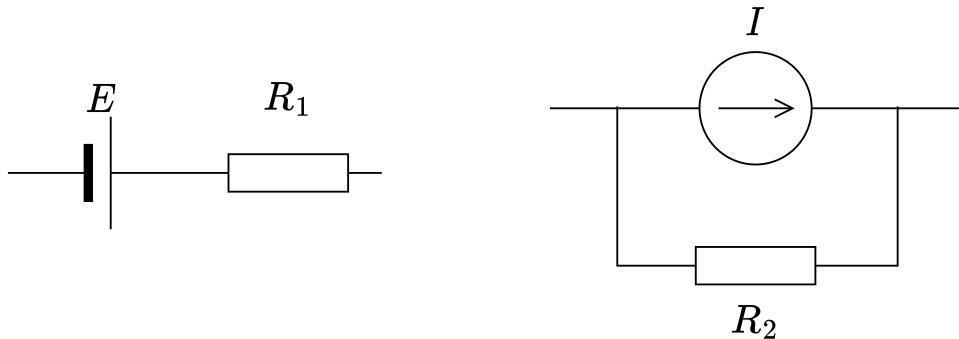


図 2.7 現実の電源の等価電圧源と等価電流源

### 2.3.8 重ね合わせの理

重ね合わせの理は、複数の電源が回路中に存在するとき、任意の地点の電流および、電圧はある 1 つの電源のみ機能していて他の電源を‘ゼロ’にしたときの値をすべての電源について重ね合わせたものに等しい、という原理である。この原理は回路に線形性があるときのみ成り立つ。電源をゼロにするとは、電圧源の電圧を 0、電流源のある枝の電流を 0 にするということである。つまり、図 2.8 のように、電圧源を短絡（ただの配線に置き換える）、電流源を開放する（配線を切り離す）ということに相当する。

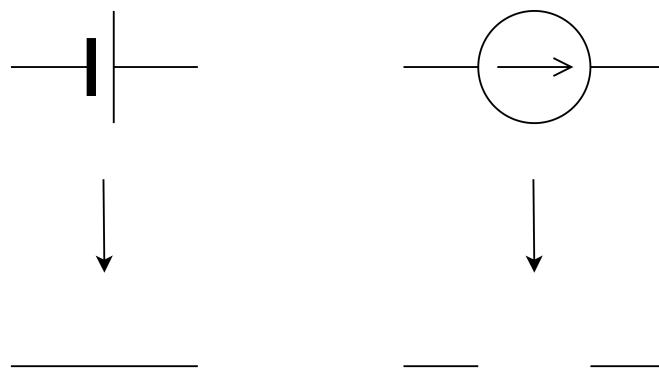


図 2.8 電圧源の短絡(左)と電流源の開放(右)

たとえば、2 つの電源として電圧源  $E_0$  と電流源  $I_0$  が存在する回路があり、ある点  $P$  での電流  $I$  が知りたいとしよう。まず、 $E_0$  の働きを知るために、 $E_0$  のみ機能をもたせた状態にし、 $I_0$  の両端を開放する。その状態での  $P$  での電流を求めるとき  $I_1$  であった、とする。

次に  $I_0$  の働きを知るために、 $I_0$  のみ機能をもたせた状態にし、 $E_0$  の両端を短絡する。その状態での  $P$  での電流を求めるとき  $I_2$  であった、とする。

これらの結果から、 $I = I_0 + I_1$  が求められる。

電源が複数あったからといってキルヒホッフの法則が使えなくなるわけではない。重ね合わせの原理は、より複雑な回路を解析するときに有用である。

### 2.3.9 テブナンの定理

テブナンの定理は、複数の素子から構成された回路網から取り出した2つの端子間を図2.9のように非常に単純な等価回路に置き換える定理である。この図における等価回路の電圧 $E$ を（2端子間を開放したときの電圧であるから）開放電圧と呼ぶ。 $R_0$ は2つの端子から内部を見込んだ内部抵抗である。2つの端子間に抵抗 $R$ が接続されているとき、この抵抗を流れる電流 $I$ は以下のように表される。

$$I = \frac{1}{R_0 + R} E$$

この定理を実際に適用する回路において、考える端子間は必ずしも開放されている必要はない。定理を適用するときのみ自分で開放して考えれば良い。内部抵抗を求めるときには、回路網側の電源について、電圧源は短絡、電流源は開放して考える。

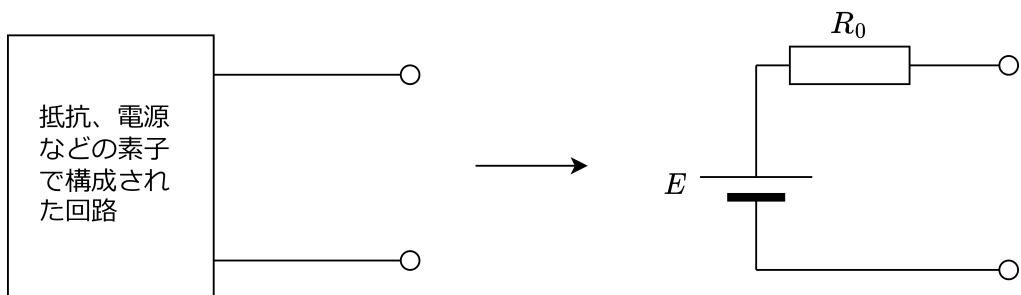


図2.9 テブナンの定理

### 2.3.10 定理の利用例

定理を活用して、以前閉路解析法で解いた回路を改めて解いてみよう。

問題

図2.10の回路において、 $I_1$ と $I_2$ を求めよ。

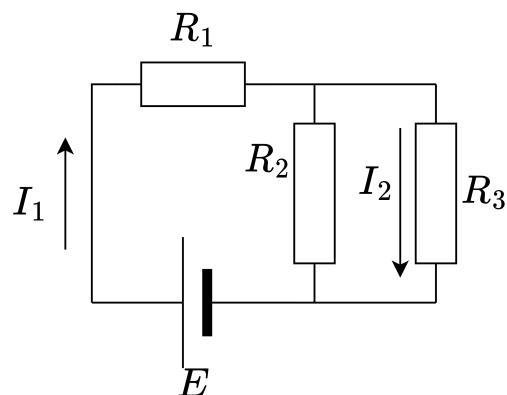


図2.10

### 分流と合成抵抗による解法

並列の抵抗を $//$ の記号で表すとする。全体の合成抵抗は  $R_1 + R_2//R_3 = R_1 + \frac{R_2R_3}{R_2+R_3} = \frac{R_1R_2+R_2R_3+R_3R_1}{R_2+R_3}$  であるから、オームの法則より

$$I_1 = \frac{R_2 + R_3}{R_1R_2 + R_2R_3 + R_3R_1} E$$

次に、分流を用いれば、

$$\begin{aligned} I_2 &= \frac{R_2}{R_2 + R_3} I_1 \\ &= \frac{R_2}{R_1R_2 + R_2R_3 + R_3R_1} E \end{aligned} .$$

### 電源の等価変換による解法

$I_1$  は工夫できないので省略する。

$I_2$  を求めよう。図 2.11 のように、電源の変換と合成を繰り返し適用すれば、最終的にオームの法則が適用できる形になり、

$$I_2 = \frac{1}{R_1//R_2 + R_3} \frac{R_2}{R_1 + R_2} E$$

よって、

$$I_2 = \frac{R_2}{R_1R_2 + R_2R_3 + R_3R_1} E .$$

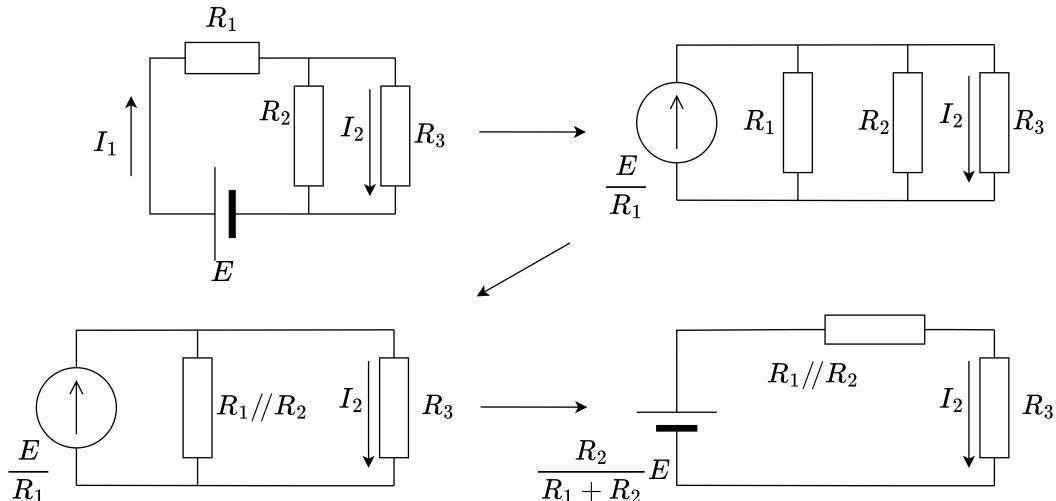


図 2.11 電源の等価変換と抵抗の合成を繰り返す過程

先にも述べたが、この方法は段階的にテブナンの定理を利用しているのと等価である。

### 2.3.11 テブナンの定理による解法

$I_1$  については、 $R_1$  の両端を開放して開放電圧と内部抵抗を考えると先に示した分流と合成抵抗による解法と全く同じ形になるため省略する。

$I_2$  について、 $R_3$  の両端を開放して、開放電圧と内部抵抗を求めよう。開放電圧  $V$  は、 $R_2$  の両端の電圧に等しい。よって、分圧を用いて

$$V = \frac{R_2}{R_1 + R_2} E$$

内部抵抗は、 $E$  を短絡して考えると、

$$\begin{aligned} R &= R_1 // R_2 \\ &= \frac{R_1 R_2}{R_1 + R_2} \end{aligned}$$

よって、 $R_3$  に流れる電流は、

$$\begin{aligned} I &= \frac{1}{R + R_3} V \\ &= \frac{R_2}{R_1 R_2 + R_2 R_3 + R_3 R_1} E \quad . \end{aligned}$$

このように種々の公式を利用することにより、キルヒホッフの法則よりも楽に答えを求めることができる。

## 2.4 正弦波交流回路の解析

### 2.4.1 正弦波交流

$$v(t) = V_m \sin(\omega t + \theta)$$

の形で表された電圧（または電流）を持つ回路を正弦波交流回路という。

$\omega$ を角周波数といい、周波数  $f$  を用いて  $\omega = 2\pi f$  と表される。 $V_m$  を振幅、 $\theta$ を位相といい、交流回路においては、振幅のほかに位相のズレまで考慮する必要が出てくる。 $\theta = 0$  のときに対し、たとえば  $\theta = -\frac{\pi}{2}$  のときは位相が  $\frac{\pi}{2}$  遅れているなどと言い、逆に  $\theta > 0$  であれば位相が進んでいると言う。

### 2.4.2 交流回路上の素子

#### 抵抗

交流回路上に抵抗がある場合について考察しよう。直流回路におけるオームの法則は  $R$  が不変ならば  $E = RI$  が成り立つことを約束するものであった。実はこの法則は、電圧や電流がが時間変化したとしても成り立つとして拡張してよい。ゆえに、抵抗の両端に正弦波の電圧を印加すれば、

$$\begin{aligned} v(t) &= Ri(t) \\ i(t) &= \frac{V_m}{R} \sin(\omega t + \theta) \end{aligned}$$

よって位相は同相であり、電圧の振幅  $V_m$ 、電流の振幅  $I_m$  に対し、

$$V_m = RI_m$$

の関係がある。

### キャパシタ

キャパシタ、またはコンデンサは、2つの導体板が向き合った構造の素子であり、両端に電圧  $V$  を印加すると、電荷移動が発生して2つの導体板がそれぞれ  $+Q$ 、 $-Q$  [C] に帯電する。このとき各導体板に蓄えられる電荷量は与えた電圧に比例し、

$$Q = CV$$

の関係をみたす。この定数  $C$  を静電容量、または単に容量といい、単位をファラッド [F] で表す。

電流の基本定義から、 $i(t) = \frac{dq(t)}{dt}$  であるから、この  $q$  に  $q(t) = Cv(t)$  を代入すれば、

$$i(t) = C \frac{dv(t)}{dt} .$$

この関係は逆に積分形式でも表され、

$$v(t) = \frac{1}{C} \int_{-\infty}^t i(t) dt .$$

キャパシタ  $C$  の両端に正弦波電圧を印加すれば、

$$\begin{aligned} v(t) &= V_m \sin \omega t \\ i(t) &= \omega CV_m \sin(\omega t + 90^\circ) \end{aligned}$$

となるから、電流の位相が電圧より常に  $90^\circ$  進み、電圧の振幅  $V_m$ 、電流の振幅  $I_m$  に対し、

$$V_m = \frac{1}{\omega C} I_m$$

が成り立つ。 $\frac{1}{\omega C}$  を容量性リアクタンスという。

### インダクタ

インダクタはコイルとも呼ばれる、鉄心、フェライトコアまたは空心にエナメル線等の導線を巻き付けた素子である。電磁気学によれば、コイルに流れた電流が時間変化すると、自己誘導により逆起電力  $v(t)$  が  $i(t)$  と逆向きに発生し<sup>\*10</sup>、以下のように表される。

$$v(t) = L \frac{di(t)}{dt}$$

この関係は以下の積分形式でも表される。

$$i(t) = \frac{1}{L} \int_{-\infty}^t v(t) dt$$

これらの式において、 $L$  をインダクタンスと呼び、ヘンリー [H] の単位を用いる。

---

<sup>\*10</sup> 電磁気学においては、電流に沿った向きに  $v = -L \frac{di}{dt}$  と、符号付きで与えられるが、はじめから電流と逆向きを正方向にとっておけばマイナスを付ける必要はない。

インダクタ  $L$  の両端に正弦波電圧を印加すれば、

$$v(t) = V_m \sin \omega t$$

$$i(t) = \frac{1}{\omega L} V_m \sin(\omega t - 90^\circ)$$

となるから、電圧の位相は電流の位相よりも常に  $90^\circ$  進んでおり、電圧の振幅  $V_m$ 、電流の振幅  $I_m$  に対し、

$$V_m = \omega L I_m$$

の関係がある。 $\omega L$  を誘導性リアクタンスという。

### 2.4.3 実効値

$$V_e = \sqrt{\frac{1}{T} \int_0^T v^2(t) dt}$$

のように 2 乗平均平方根値で定義される電圧値（または電流値）を実効値という。ただし、 $T$  は周期である。正弦波交流に対しては、

$$V_e = \frac{V_m}{\sqrt{2}}$$

$$I_e = \frac{I_m}{\sqrt{2}}$$

を満たす。一般的に 100 V の交流などというときには、実効値を示している。

### 2.4.4 交流回路の例

#### RL 回路

まず、正弦波交流電源  $e(t) = V_m \sin \omega t$  と抵抗  $R$ 、インダクタ  $L$  を直列接続した回路を考えよう。このとき回路方程式は、

$$Ri + L \frac{di}{dt} = V_m \sin \omega t$$

このとき、回路が接続された瞬間は過渡現象が生じるが、十分長い時間が経つと応答電流も正弦波となり、そこから逸脱することはない。この状態を定常状態と呼ぶ。この章では定常状態のみを考えるものとする。この式において、応答電流の一般解は微分方程式を解けば得られるが、定常状態の解のみを求めるならば、応答電流の形を  $i(t) = I_m \sin(\omega t + \theta)$  とおいて解けば良い。そのように解けば、

$$i(t) = \frac{V_m}{\sqrt{R^2 + \omega^2 L^2}} \sin(\omega t + \theta), \quad \theta = -\tan^{-1}\left(\frac{\omega L}{R}\right)$$

これを解く過程で三角関数の合成を用いた。

## RC 回路

同様に抵抗  $R$  と、キャパシタ  $C$  を直列接続した回路を考えよう。このとき回路方程式は、キャパシタにかかる電圧を  $v$  とすれば

$$Ri + v = V_m \sin \omega t$$

$$i = C \frac{dv}{dt}$$

同じように解を仮定して解けば

$$i(t) = \frac{\omega CV_m}{\sqrt{1 + \omega^2 C^2 R^2}} \sin(\omega t + \theta), \quad \theta = \tan^{-1}\left(\frac{1}{\omega CR}\right)$$

この解法は三角関数の合成の公式を使わなければいけなかったり、微分方程式をキルヒ霍フの法則を用いて立てる過程で符号のミスや連立のミスが起こりやすかったりするためあまり使い勝手がよくない。そして、直流回路で紹介した種々の定理が使えないという問題がある。ここから先は複素数を用いた方法により、直流と同じように交流回路が解けることを示していく。

### 2.4.5 複素記号法

$i$  は電流と混同しやすいため、虚数単位を  $j$  とする。オイラーの公式  $e^{j(\omega t + \theta)} = \cos(\omega t + \theta) + j \sin(\omega t + \theta)$  を用いれば、信号  $s(t) = A \sin(\omega t + \theta)$  について

$$s(t) = \text{Im}[Ae^{j(\omega t + \theta)}] = \text{Im}[Ae^{j\theta} e^{j\omega t}] \quad (2.2)$$

と表される。ただし、 $\text{Im}[X]$  は  $X$  の虚部を表す。このとき、時間に依存しない振幅と位相を抽出した成分  $S = Ae^{j\theta}$  を  $s(t)$  のフェーザ表示という。フェーザ表示は時間ではなく後述するように周波数に依存するから、周波数領域での表示ということがある。これに対して、 $s(t)$  を時間領域での表示という。時間領域で微分した信号のフェーザ表示を考えると、指数関数であるから以下のように与えられる。

$$\frac{ds(t)}{dt} \leftrightarrow j\omega S$$

すなわち、時間領域での微分は周波数領域で  $j\omega$  を掛けることに対応する。逆に、時間領域での積分は周波数領域で  $1/j\omega$  を掛けることに対応する。

抽象的でわかりにくいので、実際に回路を考えよう。先の RL 直列回路を再び考えるとする。微分方程式を再掲すれば、

$$Ri + L \frac{di}{dt} = V_m \sin \omega t$$

オイラーの公式を考えると、 $\sin(\omega t + \theta)$  は  $e^{j(\omega t + \theta)}$  の中に“含まれている”と見なせるため、 $e^{j(\omega t + \theta)}$  という形に置き換えて考えると、

$$Ri + L \frac{di}{dt} = V_m e^{j\omega t}$$

ここから  $i$  を仮定して解くが、この  $i$  も  $e^{j(\omega t + \theta)}$  で置き換えて表すと、

$$i = I_m e^{j(\omega t + \theta)}$$

$$\frac{di}{dt} = j\omega I_m e^{j(\omega t + \theta)}$$

と表されるから、微分方程式は、

$$RI_m e^{j(\omega t + \theta)} + j\omega L I_m e^{j(\omega t + \theta)} = V_m e^{j\omega t}$$

となる。ここで、よく見ると  $e^{j\omega t}$  で割ることができるから、全体を  $e^{j\omega t}$  で割ると、

$$RI_m e^{j\theta} + j\omega L I_m e^{j\theta} = V_m$$

というように時間依存性が消えた式となる（これがフェーザ表示で時間依存性を考える必要性がない理由である）。ここで、 $I_m e^{j\theta} = \dot{I}$  とおけば<sup>11</sup>、 $\dot{I}$  こそが、 $i(t)$  のフェーザ表示であり、

$$\begin{aligned} R\dot{I} + j\omega L\dot{I} &= V_m \\ \dot{I} &= \frac{V_m}{R + j\omega L} \\ &= \frac{V_m}{\sqrt{R^2 + \omega L^2} e^{j\theta_0}} \\ &= \frac{V_m}{\sqrt{R^2 + \omega L^2}} e^{-j\theta_0} \end{aligned} \tag{2.3}$$

ただし、 $\theta_0 = \tan^{-1} \omega L / R$  である。ここで、フェーザ表示の定義式 (2.2) を思い出せば、 $\dot{I}$  の時間領域信号  $i(t)$  は以下のように逆変換できる。

$$i(t) = \frac{V_m}{\sqrt{R^2 + \omega L^2}} \sin(\omega t - \theta_0)$$

これは、時間領域のまま解くのと同じ結果である。この過程で、前半は時間領域が消えることを示しただけなので、実際に問題を解くときにはフェーザ表示を置いた式 (2.3) から始めれば良い。電流と電圧のフェーザ表示を複素電流、複素電圧などとよぶ。この式では、微分要素が消え、抵抗  $R$  のほかに、あたかもインダクタが  $j\omega L$  の抵抗であるかのように振る舞っていることがわかる。一般に周波数領域であたかも抵抗のように振る舞っている複素数  $\dot{Z} = R + jX$  をインピーダンスと呼ぶ。 $R$  は抵抗、 $X$  はリアクタンスである。この回路では、合成インピーダンスは  $R + j\omega L$  である。キャパシタについても同じように解析すれば、インピーダンスは  $\frac{1}{j\omega C}$  であることがわかる。インピーダンスを用いることで、インダクタやキャパシタが含まれている交流回路であってもあたかも複素数の抵抗を持った直流回路のように解くことができる。これにより、これまでに紹介したすべての解析手法が再び使えるようになった。

確認のため、この手法で RC 直列回路を解いてみよう。キャパシタのインピーダンスは  $1/j\omega C$  であり、電圧のフェーザ表示  $\dot{V} = V_m e^{j0} = V_m$  を用いると、複素電流  $\dot{I}$  は、

$$\begin{aligned} \dot{I} &= \frac{1}{R + 1/j\omega C} \dot{V} \\ &= \frac{j\omega C}{1 + j\omega CR} \dot{V} \\ &= \frac{\omega C \dot{V}}{\sqrt{1^2 + (\omega CR)^2}} e^{j(\frac{\pi}{2} - \theta)}, \quad \text{where } \theta = \tan^{-1} \omega CR \end{aligned}$$

---

<sup>11</sup> フェーザ表示は複素数であり、ベクトルに近い性質をもっているからただのスカラと区別するためにドットをつけた記法で示すことが多い。個人的には時間微分のニュートン記法と混同しそうなのであまりよい記法とは思わないが、慣習に従っておくことにする。

と表される。ここで、 $\tan(\frac{\pi}{2} - \theta) = 1/\tan\theta$  であるから、 $\frac{\pi}{2} - \theta$  を改めて  $\theta$  とおけば、 $\theta = \tan^{-1} 1/\omega CR$  である。最後に時間領域に戻せば

$$i(t) = \frac{\omega CV_m}{\sqrt{1 + (\omega CR)^2}} \sin(\omega t + \theta) \quad .$$

非常に楽に解くことができた。

## 第3章

# ハードウェア設計

### 3.1 ハードウェア概観

本章ではハードウェア設計について説明する。今回設計したハードウェアは大きく分けてベース、光センサ基板、メイン基板、その他にユニバーサル基板を用いた小基盤から実装されている。

外観を示すと、図 3.1 のようになっている。

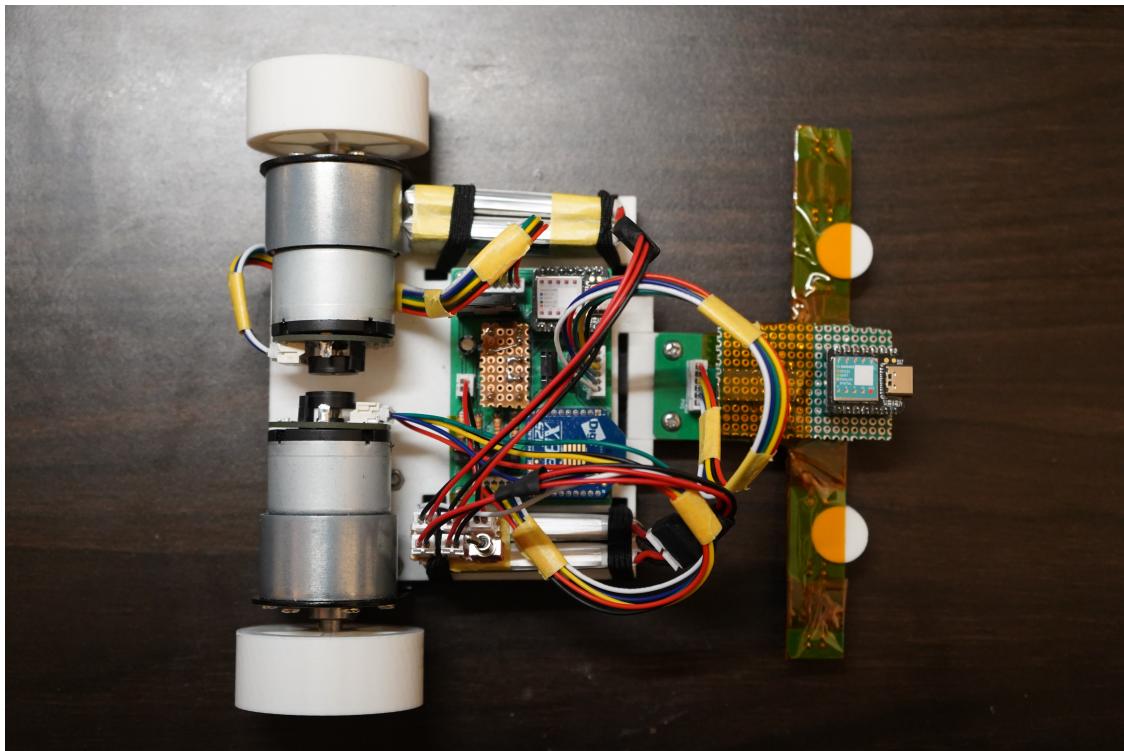


図 3.1 ハードウェアの外観

### 3.2 回路図

図 3.2 に回路図を示す。

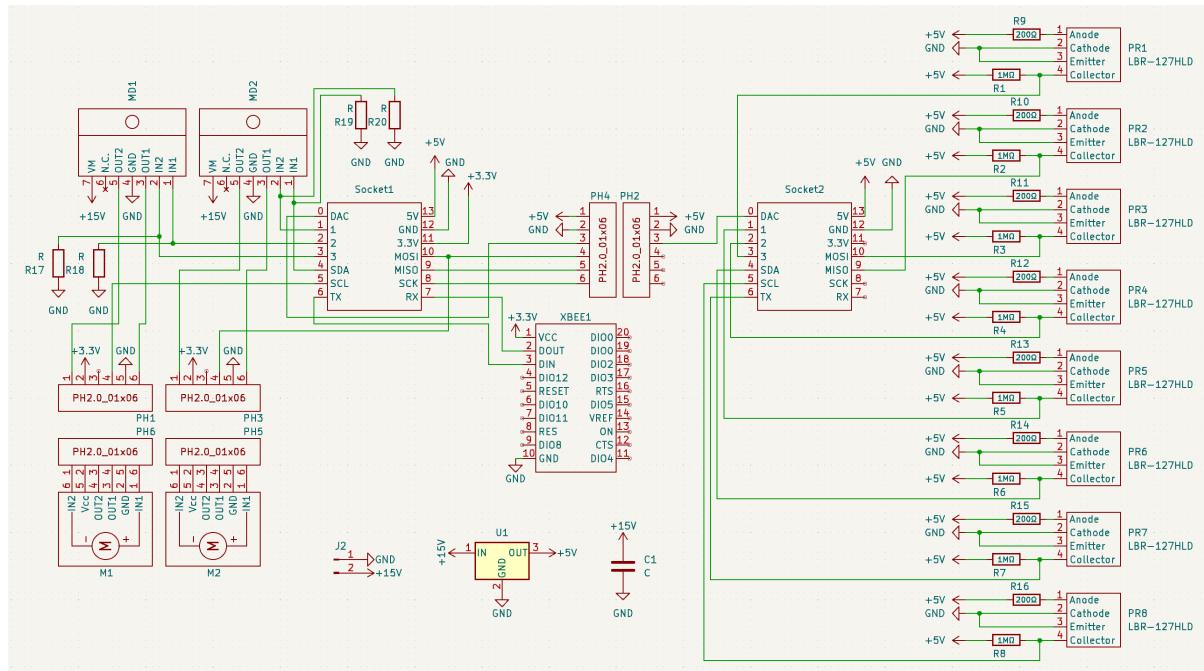


図 3.2 回路図

PH2 に接続された部分が光センサの基板部分の回路であり、それより左側がメイン基板の回路である。

### 3.3 部品表

以下に部品表一覧を示す。モータについては、12 V、ブラケットありの製品で、530RPM（望月氏出資機）、1000RPM（筆者出資機）のものを使用した。

記号	部品名	単価(円)	個数	購入先
(センサ部)				
PR	フォトリフレクタ	70	8	秋月電子通商
R1 ~ R8	チップ抵抗 (1 MΩ)	0.71	8	AliExpress
R9 ~ R16	チップ抵抗 (200 Ω)	0.71	8	AliExpress
MP1	IC ソケット (16P)	13	1	秋月電子通商
Socket2	IC ソケット (16P)	13	1	秋月電子通商
PH	PH コネクタ ベース付ポスト	15	1	秋月電子通商
ユニバーサル基板に実装	Seeeduino Xiao	850	1	秋月電子通商
(メイン基板)				
ピンソケットに実装	Seeeduino Xiao	850	1	秋月電子通商
MD1,MD2	モータドライバ TB6643KQ	350	2	秋月電子通商
R1 ~ R4	カーボン抵抗 (炭素皮膜抵抗) 1/4W 1 kΩ	1	4	秋月電子通商
Socket1	IC ソケット (16P)	13	1	秋月電子通商
PH	PH コネクタ ベース付ポスト	15	3	秋月電子通商
C1	電解コンデンサ (100 μF 25 V)	10	1	秋月電子通商
(その他)				
機体に実装	1セルリチウムイオンバッテリー	321	4	Banggood
機体に実装	エンコーダ・ギアボックス付きモータ	1805	2	AliExpress

表 3.1 部品表

### 3.4 ベース

ベースは 3D プリンタを用いて造形されており、材質は PLA フィラメントである。部品としては本体、センサ部連結パーツ、接地部、その他小部品に大別される。図 3.3 に 3D モデルを示す。

### 3.5 光センサ部

光センサ部は、プルアップ抵抗とフォトトランジスタによる光センサが 8 個付いており、これらの信号はセンサ用マイコンに送られてソフトウェア的に 1 つの信号に集約されたうえでメイン基板にアナログ値として送られる。

### 3.6 光センサ拡張マイコン基板

本来は光センサ出力はマルチプレクサを経由してメイン基板すべて処理される予定であったが、問題が発生したため、マルチプレクサの部分からユニバーサル基板を生やし、その基板上に別のマイコンを載せて別処理させることになった。前方にマイコン接続用の USB Type-C が飛び出しているため、正面衝突に弱く、拡張マイコンごと外れることもあるため、取り扱いに注意しなければならなかった。

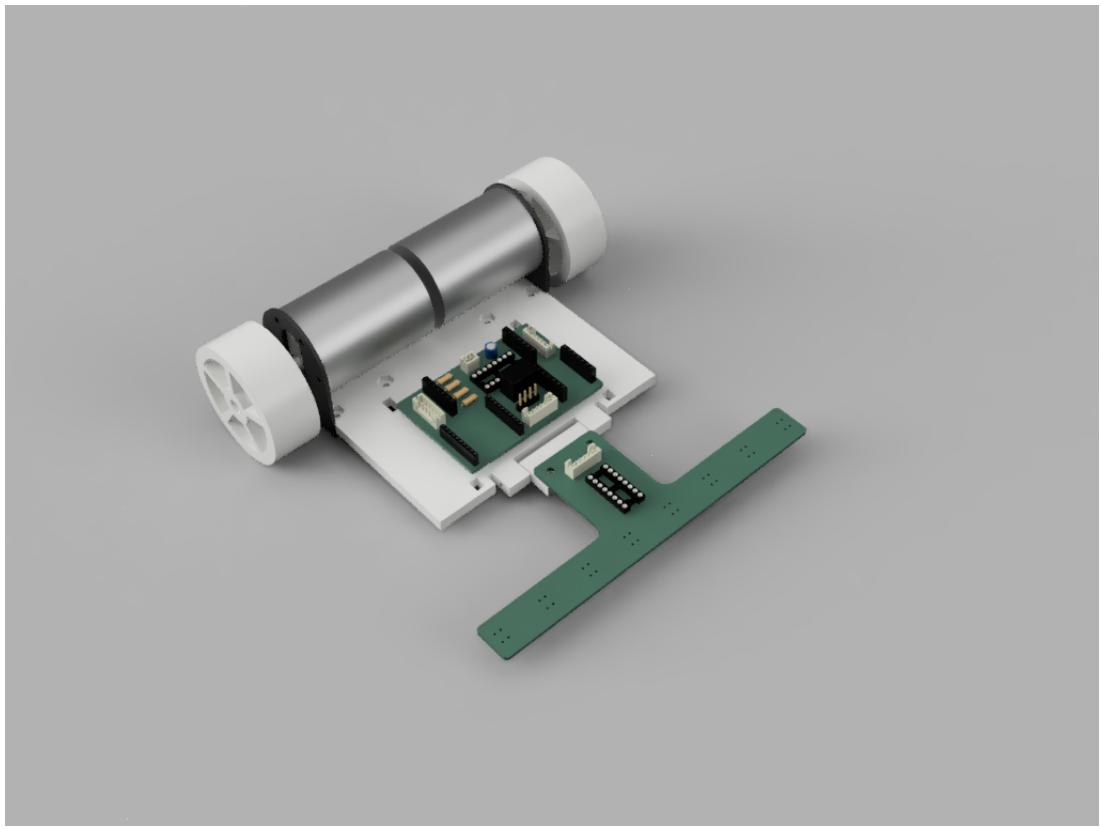


図 3.3 ハードウェア設計の 3D モデル

### 3.7 メイン基板

メイン基板は、光センサ、エンコーダなどすべてのセンサ値を集約し、モータを適切に制御する役割がある。基板上の役割は電源回路と信号回路に大別される。

電源回路としては、公称値 3.8 V のバッテリを 4 セル直列接続した約 16 V の電源と、レギュレータによって 5 V に降圧された電源の 2 種類の電圧が運用されている。16 V 電源はモータを回すために使用し、5 V 電源はマイコンやセンサを作動させるために使用されている。元電源側にコンデンサを並列に挿入してあるため、瞬間的な電圧変動にもある程度の耐性をもつ。

信号回路は、センサ値の取得、マイコンによる制御、モータの PWM 制御等に使用されている。

### 3.8 電源基板

バッテリセルを 4 つ直列接続するための PH ソケットが並んだユニバーサル基板。すべての電源はこの基板から取られている。スイッチもこの基板に実装されている。

## 第4章

# ソフトウェア設計

### 4.1 ソフトウェア概観

#### 4.1.1 全体構成

本章ではソフトウェアの構成に必要な理論について説明する。今回構成した走行体には、光センサとモータのエンコーダという2つのセンサが搭載されており、それらの値をモータの制御に反映したフィードバック制御アルゴリズムを構築する必要がある。そこで本機ではモータの回転速度を制御量、PWM制御の入力量を操作量としたPID制御を用いることとした。ただし、一般的にPID制御は1入力1出力の系であるから、光センサとエンコーダという2つのセンサ値のある系ではそのまま適用することができず、少し工夫が必要となる。ライントレーサの制御には例えればETロボコンについての記事[4]が詳しい。これによれば、光センサのPID制御値とエンコーダのPID制御値をそれぞれ計算し、和を取ったものを直接操作量としてPWM制御に入力すればよいとされている。この手法は両方の制御がダイレクトに操作量に反映されるのでよいパラメータを設定できれば非常に応答性が高いプログラムになると思われる。しかし、6つのパラメータを同時に調節する必要があり、走行の様子からどのパラメータをどの程度調節すればよいのかを見極めるのは初学者には難しい。そのため、今回はセンサ値のPIDから各モータの目標速度を変更し、その速度に向かってモータ側のPID制御を行うという2段のPID制御を行う方式を最終的に採用した。この方式は、センサ値から直接操作量を変更しないで間にモータ側のPID制御を挟むため、必ずしも最適な速度制御とはならない可能性、およびディレイがある可能性があることが欠点である。しかし、モータ側のPIDパラメータの調整とセンサ側の調整を切り離し、1度に3つのパラメータのみ対象として調整することができる事がメリットである。この構成を1つのモータに注目してブロック線図で示すと、以下のようなになる。

PID制御器が2つ直列に接続されていること、および速度のPID制御の前で3つの値の加減算が行われていることが特徴的である。

### 4.2 光センサ値の検出

#### 4.2.1 コース線の中心の推定

先に述べたように、本機は8つのセンサを積んでいるが、PID制御に回すためには1つのスカラ値に変換する必要がある。センサ値の統合には様々な方法があるが、線が機体の外側に行くほど絶対値の大きい値を返すような単調な関数となっていることが望まれる。そこで、本機では2つのモータを結んだ線の中心から、コー

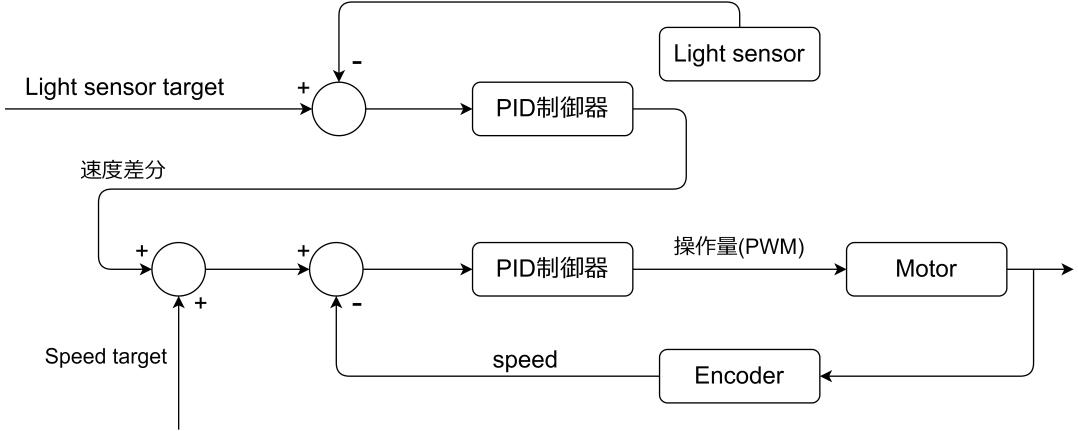


図 4.1 採用したシステムのブロック線図

ス線の中心と推定される位置の角度をセンサ値として利用する。ここで、コース線が左側にあるとき、負の値を返し、右側にあるときに正の値を返すとする。位置関係を示すと、図 4.2 のようになる。

まず、コース線の中心位置を推定する方法を考えよう。図の左側のセンサから順に 0 番から 7 番であるとする。車体の中心線 car\_censor を基準として右側を正、左側を負とする座標系を考えると、各センサの位置は 0 番から順に -7, -5, -3, -1, 1, 3, 5, 7 [cm] にある。つまり、 $i$  番目のセンサ位置  $P_i$  は

$$P_i = -7 + 2i$$

である。次に、センサ  $i$  がコース線に乗っているか判定する関数を  $b[i]$  とすれば、

$$b[i] = \begin{cases} 1 & (\text{センサ } i \text{ がコース線上にある}) \\ 0 & (\text{センサ } i \text{ がコース線上にない}) \end{cases}$$

$b[i]$  はセンサの実測値に対してしきい値を付けることで容易に実装することができる。

よって、これらを足し合わせ、“コース線上に乗っている” センサ数  $N$  で割り、平均を取ることでコース線の推定位置  $P_e$  は

$$P_e = \frac{1}{N} \sum_{n=0}^7 b[n] P_n$$

#### 4.2.2 角度の推定

次に、車軸から見込んだ推定角度  $\theta_e$  を求めよう。本機において、車軸とセンサのある位置までの距離  $d_{base} = 13[\text{cm}]$  である。また、 $\tan \theta_e = \frac{P_e}{d_{base}}$  である。ゆえに、

$$\theta_e = \tan^{-1} \frac{P_e}{d_{base}}$$

である。

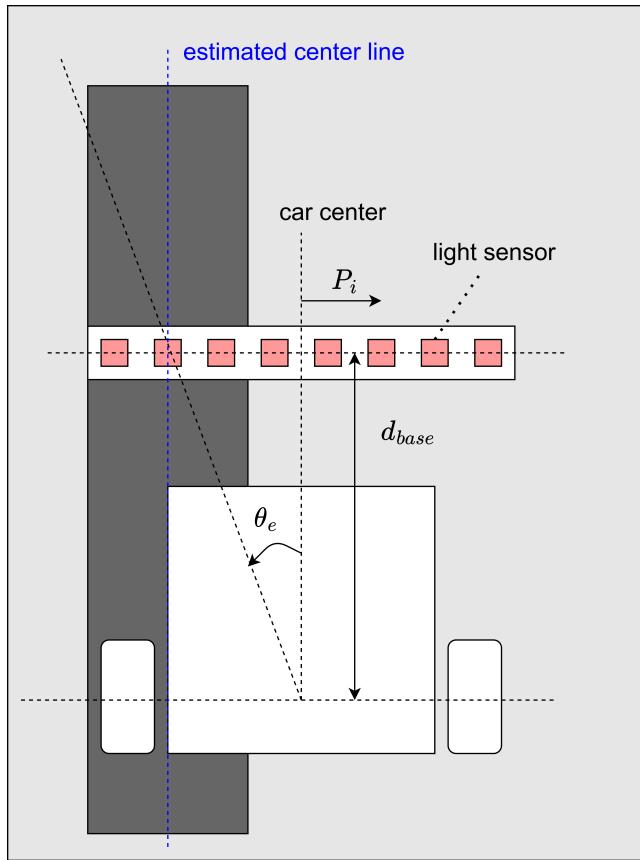


図 4.2 位置関係

三角関数は計算量が他の演算に対して大きいのでボトルネックとなることがある。その際には、 $\tan^{-1}$  をテイラー展開して、適当な次数で切ることで幕関数として扱えば良い<sup>\*1</sup>。今回の配置であれば、5次近似程度あれば十分な精度である。しかし、実際に実験した結果、今回は特にボトルネックにならなかったので、検討はしたもの、特に必要はなかった。

#### 4.2.3 読み込みのボトルネック

以上で 8 つのセンサを 1 つのスカラ値に変換することができた。この処理では 8 つのセンサを順次読みこむことから、ボトルネックとなり得ることに注意する必要がある。実は当初設計では採用したマイコンではピンが不足していたのでマルチプレクサを順次切り替えて処理する形式を取っていた。しかし、実際に試したところ、回路設計の関係なのか、1 度切り替えてから電圧が安定するまで  $110 \mu\text{s}$  程度要することがわかった<sup>\*2</sup>。このディレイは毎回挟むと後述の速度検出のサンプリング周波数が不足するため解消する必要があった。そのため、本機ではマイコンを 2 台利用するように設計を変更し、光センサ値の検出に関する部分をすべて移動

<sup>\*1</sup>  $\arctan x$  のテイラー展開は 2 次近似まで  $x$  であるから、大きさの正規化さえ気をつければ  $P_e$  を直接角度の推定値として扱ってもあまり問題はない。今回は精度を優先した。

<sup>\*2</sup> マルチプレクサ側の問題ではない。モーターのエンコーダも同様にマルチプレクサを使っていたが、 $3 \mu\text{s}$  程度のディレイで問題なく動作した。

した。遅延への懸念から、光センサ用マイコンから本マイコンへの通信には通信プロトコルを使用せず、適切なレンジにマッピングしたアナログ値(電圧)を出力し、受け取り側で扱いやすい値に再マッピングすることで問題なく利用することができる。今回は受け取り側で [-10, 10] の範囲に再マッピングした。

## 4.3 速度の検出

### 4.3.1 エンコーダ

速度の検出にはモータに付いているエンコーダを活用する。エンコーダにはホールセンサが付いており、車輪を回転させると、1回転で決まった回数  $P$ だけデジタルでパルスが送られる。本機においては  $P = 11$  である。つまり、信号を  $f(t)$  とし、モータを定速で回転させたならば  $f(t) = 1$  (HIGH) となる時間と、 $f(t) = 0$  (LOW) となる時間を周期的に繰り返す。通常モータは正転と逆転を判別するために位相の異なるホールセンサが2つ搭載されている。しかし、今回はモータは逆転しないものとし、1つのエンコーダにのみ着目する。

### 4.3.2 センサ値のサンプリング

デジタル値が流れている生データから速さを取得するためには、信号の値の変化を捉えてその時間間隔を調べればよい。1つのエンコーダで捉えられる値の変化は HIGH to LOW または LOW to HIGH の2種類である。仮に両方の変化を捉えるとすれば、1回転で  $2P$  回の変化を捉えることとなる。一般に、より多くの変化を捉えるほうが精度のよい速さが求められるが、サンプリング周波数に注意する必要がある。プログラム中で速度を毎ループ計算せずに非同期で処理する場合、1ループにかかる時間の逆数がサンプリング周波数である。そして、1ループで1回の変化が起こる状態(毎回のループで HIGH -> LOW -> HIGH と変化している状態)が取得しうる最高速度であり、それ以上の速度は測定できない。しかし、そもそもナイキスト定理を考えれば、この状態はすでにエイリアシングが発生して精度が低下しているから元の信号の周波数は更に低くなければいけない。さらに言えば、ナイキスト定理では2倍のサンプリング周波数があれば良いことになっているが、波形を復元する特別な工夫がないなら10倍以上の周波数を使うべきである。フーリエ変換を考えて厳密に言えば、デジタル信号は矩形波であるから、無限大の周波数がなければ修復できない。そのため、どのようなサンプリング周波数で計測しても、たまに飛び値が出る。しかし、モータは瞬間に追従することはできないうえ、次の瞬間には正しい値に戻っていることを考えれば飛び値が出ることは問題にならない。そして、LOW->HIGH->LOW をあたかも1つの波であるかのように捉えてその周波数を考えてみればこの計測法でも比較的正しい速度が計算できることがわかる。こちらで考えても周波数が高すぎるときは、変化の読み落としが発生しているので全く信用にならない。本機では LOW to HIGH のみを捉えることとする。

ここまででは、非同期で処理する場合を考えた。では同期処理ではどのような問題があるだろうか。まず、同期処理では毎回の処理でそのときの速度を定める。つまり、毎回速度が確定するまでこの処理のみを実行するので、読み落としは発生せず、エイリアシングの問題等は一切考える必要がない。ここだけ聞くと、同期処理のほうがよく聞こえるかもしれないが、問題はこの処理が終わる (= エンコーダの変化を2点で捉える) まで次の処理に進めないことである。高速走行時には短い時間で変化を捉えられるので大きな問題にはならないかもしれないが、低速時には待ち時間が増えるし、停止時には停止しているかどうかを明確に判別できるだけの最大待ち時間待たなければならなくなる。その時間はモータに依存するが本機においては最低でも 20ms であった。つまり、1ループに最大 20ms、高速走行時でも非同期処理よりは当然遅く、数 ms の待ち時間が発生

する。待ち時間とはつまり制御が入らずに空走する時間である。例えば、50cm/sで走行中に5msの待ち時間があったとすると、空走距離は  $50\text{cm/s} \times 5\text{ms} = 0.25\text{cm}$  である。僅かな量に感じるかもしれないが、これだけの空走距離があると急なカーブ等には対応することが困難である。つまり、同期処理で速度を取得すると非常に大きなボトルネックになるので、速度の計算は非同期処理にする必要がある。

#### 4.3.3 生データから速さへの変換

前節でモータ1回転に捉える変化数を決めた。改めて  $P$  とおく。つまり、本機においては  $P = 11$  である。次に、値の変化にかかった時間を  $\Delta t [\mu\text{s}]$  とする。モータのギア比(減速比)を  $R$  とする。タイヤの直径を  $D [\text{cm}]$  とすれば、タイヤ外周部の速度  $v [\text{cm/s}]$  は次のように表される。

$$v = \frac{\pi D \times 10^6}{R P \Delta t}$$

これは実質的に時間の1変数関数であり、残りは機体に固有の定数であるから、切り分けて前処理に回すことで処理を高速化できることに留意したい。また、車体が停止しているとき、 $\Delta t$  は永遠に求まらないため、同期処理と同様に一定時間経過したら速度を0と判定するしきい値時間を用意する必要があることに注意する。

#### 4.3.4 非同期処理の実装

本機は Arduino 互換ボードを使用しており、ソフトウェアの実装言語は Arduino 言語である。そのため、Python 等高機能言語に実装されている `async/await` 等の高級な非同期処理インターフェースはない。そのような状況でシンプルに非同期処理を実装するためには、時間を確認することが有効である。Arduino 言語には、経過ミリ秒を表示する `millis()` やマイクロ秒を表示する `micros()` が存在するため、それを用いることで素朴な非同期処理を実装することができる。

その他の選択肢としては、今回は採用していないが割り込み処理にする方法が考えられる。ボード依存であるが、Arduino は割り込み処理をサポートしている。ホールセンサの信号の変化をトリガとした割り込み処理をすればより無駄のない速度計測ができる可能性がある。

#### 4.3.5 ノイズ対策

エンコーダでは、ホールセンサの信号を HIGH と LOW で送信する。回転の途中で値の境界付近の状況にあるとき、短い周期で信号が震えることがある。これはスイッチにおけるチャタリングに近い現象であり、同様に対策が必要となる。本機では、信号値を取得する際に毎ループで続けて3回取得し、すべての信号値が揃わなければ信用できないとして信号を取得し直す、というロジックによりこの問題へ対処した。

### 4.4 モータ制御

#### 4.4.1 センサの PID 制御

これまででそれぞれのセンサ値を取ることができた。次にこれをモータの動作に落とし込む必要がある。用いる PID 制御を時間の式で示せば、

$$y = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

ただし  $e$  は目標値から現在値を引いた偏差である。これはアナログ値であるから、デジタル値による記法に変換する必要がある。これにはいくつかの流儀があるが、今回は最も基本的な方式として、微小時間は無視する、積分要素は面積を取らずに現在差分を単純に足す、微分要素は差分要素に置き換えるという方法を取った。

今一度 PID 制御器の結果がどのように使われるか確認すると、光センサについての PID の結果は曲がる方向に応じてモータの目標速度を変化させる役目をもつ。モータについての PID の結果はもちろん所望の目標速度に近づくようにモータへ送る操作量を調整する。

図 4.1 にブロック線図を示してあるから、あと必要なことはそのまま実装して、パラメータの調節を行うことのみ（これが一番大変かもしれない！）である。

#### 4.4.2 光センサにおける PID 調整前のパラメータ推定

PID 制御のパラメータ調節は地道な作業であるが、論理的に考えられる部分については、ある程度あたりをつけてから始めると楽である。特に、PID 制御の調節は P 制御から始め、 $K_p$  の大きさによって他の値のレンジの決定されるので、 $K_p$  の正当性が最も重要である。ここでは、光センサの P 制御値について考察を行うことでパラメータにあたりをつけてみよう。

ライントレースを行ううえで一番走行が難しいのは急カーブである。逆にいえば、急カーブを走行してもコースアウトせずに即座に車体の安定を取り戻せる状態なら他の場所は問題にならない。では  $K_p$  をいったいどのくらいの大きさにすれば最も急なカーブを曲がり切れるだろうか？

最も急なカーブの曲率半径を  $r[\text{cm}]$  とする。車体の中心線からタイヤの中心までの距離を  $d[\text{cm}]$  とする。与えた走行速度目標を  $v_t[\text{cm}/\text{s}]$  とする。ここで、走行速度目標とはカーブに差し掛かって光センサ値による補正がかかった値ではなく、光センサ値によらない（つまり、直進中の値とも言える）である。ある瞬間、走行体は角速度  $\dot{\theta}[\text{rad}/\text{s}]$  で円運動をしているとしよう。このとき、機体の制御により、外側の車輪の目標速度は  $v_t + \Delta v$ 、内側の車輪の速度は  $v_t - \Delta v$  に変更される。ただし、 $\Delta v$  は  $K_p$  に依存する値である。このとき、速度に関する次の 2 つの式が成り立つ。

$$\begin{aligned} v_t + \Delta v &= (r + d)\dot{\theta} \\ v_t - \Delta v &= (r - d)\dot{\theta} \end{aligned}$$

ここから  $\dot{\theta}$  を削除することにより、

$$\begin{aligned} \frac{v_t + \Delta v}{v_t - \Delta v} &= \frac{r + d}{r - d} \\ &= k \end{aligned}$$

とおくと、 $k$  は  $r$  と  $d$  によって決まるからコースと車体のプラットフォームが変わらない限り定数である。この  $k$  を用いると、

$$\Delta v = \frac{k-1}{k+1} v_t$$

と表される。つまり、光センサの PID 制御全体で  $\Delta v$  より大きい速度の変化分を出せないと曲がりきれないことが実験しなくてもわかる。

本機体について考えてみよう。本機体において、 $d = 7.5[\text{cm}]$  速度  $v_t = 100[\text{cm}/\text{s}]$  で、曲率半径  $r = 10[\text{cm}]$  の円を曲がるとする。このとき、 $k = 7$  であるから、 $\Delta v = \frac{6}{8} v_t = \frac{3}{4} \times 100 = 75$  である。いま、光センサ値

が  $[-10, 10]$  の範囲でマッピングされている。現在の光センサの値を  $l$  とする。光センサの目標値は中心である 0 とすれば  $e = l$ 。ここでの偏差は定義とは反対であるが、その後のデータの与え方を反転させれば問題ない。絶対値で扱っていると考えても良い。P 制御しか行わない状態を仮定すると、 $\Delta v = K_p|l| > 75$  となる。光センサが最大値のときにぎりぎり曲がりきれると仮定すると、 $l = \pm 10$  で、 $K_p > 7.5$  となる。実際にはもっと余裕を持ちたいのでより大きな値となると推定される。たとえば、 $l = 8$  で曲がりきりたいとすると、 $K_p > 9.38$  程度となる。これと D 制御を組み合わせることで、急なカーブでもより安定して走行することができる。

実際に採用した値が  $K_p = 10.0, K_d = 12.5, K_i = 0.0002$  であることを考えるとこの考察は非常に有益であることがわかる。

#### 4.4.3 アドホックな調整

ここまでで、カーブを含めたほとんどの場合において安定した走行をするための議論は完了した。しかし、まだ特殊な場合で安定しない可能性が残されている。本節ではそのような特定の状況に対応するアドホックな処理について議論する。

安定しない可能性があるのは速度 0 からの立ち上がりの状況である。理由は 2 つある。

1 つ目は、目標速度を大きくすればするほど、初めの P 制御が強く効くからである。理想的状況であれば、速やかに目標速度に到達するため問題ない。しかし、左右のモータの回りやすさに差がある場合、初めの制御量が大きく入ると左右差が大きく出やすく、場合によっては修正する間もなくコースアウトしてしまう。

これに対応するためには立ち上がりの偏差に制限をかけなければよい。つまり、両輪が低速時に限り、現在速度から逆算したターゲット速度を与えることで実現される。これは、数式モデルとしては異なるが、一次遅れ要素に似た性質をもつ。つまり、以下のようにすればよい。

```
DelayAccelerationSpeed(targetSpeed, maxError, leftCurrentSpeed, RightCurrentSpeed)
    if leftCurrentSpeed < threshold and rightCurrentSpeed < threshold
        newTargetL = min(leftCurrentSpeed + maxError, targetSpeed)
        newTargetR = min(RightCurrentSpeed + maxError, targetSpeed)
        return newTargetL, newTargetR
    else
        return TargetSpeed, TargetSpeed
```

副次的效果として、偏差の最大値が部分的ではあるが制限されることになるため、モータ側の PID 制御の  $K_p$  をより大きくしても問題が起こりにくくなる。すべての場合で偏差の最大値を制限すれば偏差が  $maxError$  以下であることが保証されるのでそれを念頭に置いた設計ができるが、急カーブを曲がりにくくなるという難点があるため推奨しない。

安定しないもう 1 つの理由は、センサの PID 制御値の速度依存性である。4.4.2 節で速度に応じて与えるべきセンサの  $K_p$  が変化することを述べた。もちろん  $K_p$  は目標速度に合わせて作られているから、立ち上がりの速度ではいささか強く効きすぎる。そして、先に述べたように、モータには個体差があるから同じように操作量を与えてもまっすぐに進み出すわけではない。その結果、低速でラインを跨ごうとし、一方の車輪が完全に止まろうとする、逆向きに曲がろうとしても一方の車輪が完全に止まろうとする、という動作を繰り返して、振動的な挙動をしながら加速する。このような挙動ではスムーズな立ち上がりとは言えないから、対処したほうがよい。

これに対応する解決策も単純であり、両輪が低速であるときにはセンサの PID の係数を書き換えれば良い。もちろんなめらかに変化させることもできるが、低速用の係数を用意して、書き換えるか、書き換えないかという 2 段階があれば十分であると思われる。本機の場合は、50cm/s にチューニングした PID 値を用意して二段階にしておけば十分安定した走り出しどなった。

#### 4.4.4 Xbee との通信

低速で走行実験を行う場合、万が一コースアウトしても手で止めに行けば問題にならない。しかし、100cm/s といった高速で走行する場合、手で止めると機体にダメージが入ったり、手で止める前に壁に衝突する危険性がある。走行会の前に事故によって機体が破損するといった事態は絶対に避けなければならない。これを避けるため、遠隔で発進と緊急停止を行うことができる機能があるとよい。やり方は様々あると考えられるが、本機では Xbee を搭載して遠隔シリアル通信による発進と停止の機能を実装した。

### 4.5 ソースコード

長くなるので、GitHub のリポジトリと、付録 C に添付している。

GitHub リポジトリのリンクは <https://github.com/RiceCake1/LineFollow/tree/main>.

この節では、実装を関数別に簡単に解説する。なお、Arduino ではデフォルトで setup 関数と loop 関数が必要である。setup 関数は起動時に 1 度だけ呼ばれる関数であり、ピンの設定や、初期化等を書く場所である。loop 関数は setup 関数のあとに常に呼ばれる関数であり、メインループとなっている。

#### 4.5.1 光センサ関係 (光センサ用ボード側)

```
float ReadSensorBinary(int sensorNum)
```

センサがコース線に乗っているか判定する関数。

```
float ReadSensor(int sensorNum)
```

アナログ値を読み [0,1] にマップして返す関数。デバッグ用。

```
int getLinePos()
```

以前使われていた、センサ値の統合アルゴリズム v1。

```
float getLinePosThetaWeighted()
```

以前使われていた、センサ値の統合アルゴリズム v2。角度を考慮し始めた。試行錯誤の痕跡。

```
float getThetaWithArctan()
```

最終的に採用された、センサ値の統合アルゴリズム v3。解説した通り単調な関数となっていて、非常に安定性が高い。これに変更してからカーブ後の振動の収束が有意に早くなった。

#### 4.5.2 光センサ関係 (メインボード側)

```
float getLinePos()
```

メインボード側で光センサ処理用マイコンから送られてきた値を読み、[-10.0, 10.0] の範囲に再マップする関数。

```
float getPIDsensorValue(int linePos)
```

光センサの PID 制御値を計算し、返す関数。低速時に係数を変更するアドホック処理も含まれている。

#### 4.5.3 エンコーダ関係

```
int _smoothEncoderSignal(int encoderNum)
```

ノイズ対策として 3 回エンコーダの信号を読み、すべて一致するまで繰り返す内部関数。

```
bool _isChanged(int encoderNum)
```

信号が変化したかを判定する内部関数。

```
float _getSpeedWithTime(int motorNum)
```

1 つのモータについて非同期処理で速度を取得する内部関数。

```
void getSpeedWithTime(float *spd)
```

上の関数を用いて両輪の処理をまとめた関数。

#### 4.5.4 モータの制御関係

```
void controlMotor(float Rspeed, float Lspeed)
```

[-1,1] の値の 2 引数を取り、両輪の PWM 制御を簡単に行う関数。負の値ならば後退、正の値ならば前進。本番環境では内部関数だが、デバッグ用途ではそのまま使える。

```
void delayAccelerationSpeed(float targetSpeed, float *newTargetSpeed, float maxError = 10)
```

アドホック処理のための関数。立ち上がりの速度偏差に制限をかける。

```
void controlEachMotorWithPID(float rightTargetSpeed, float leftTargetSpeed)
```

左右モータの目標速度をそれぞれ引数に取り、PID 制御値を計算して、controlMotor でモータを制御する。

```
float limitSpeedOnCorner(float targetSpeed, float deacceleratedSpeed, int linePosition)
```

終盤に作られた失敗作。コーナーでは両輪とも減速すれば平常時の目標速度が速くても曲がり切れるのではないだろうか、という仮説に基づいて作られたアドホック制御関数。実際には、この関数が働き始めるときにはすでにブレーキには遅すぎる地点であることがわかった。消されたものも多いが、このプロジェクトにはこのような不採用になった関数が山程存在し、その数だけ試行錯誤があった。参考までに、最終形である現在で

はメインボードのコードは 280 行程度であるが、一時期は 500 行以上あった（1月 20 日のコミット）。

`void controlMotorFromSensors(float targetSpeed, float deacceleratedSpeed=10)`

最終的に採用されたすべてが集約された関数。センサ値の取得、光センサの PID 制御値の計算、モータの PID 値の計算、モータへの操作量の入力までがすべてこの関数で繋がった。

#### 4.5.5 その他

`float mapFloat(float inputValue, float inputLower, float inputUpper, float outputLower, float outputUpper)`

ある範囲内の実数を別の範囲にマッピングするための関数。

`show...()`

show 関数系は名前が示すものをシリアルで表示するデバッグ用関数。

`initAll()`

名前の通り、変数の初期化等をまとめている関数。setup で実行するほか、Xbee で停止させていた走行体を再び走行させるときに実行される。

`Xbee()`

無線通信を行い、ソフトウェア的に停止、発進を行うための関数。高速でのデバッグでは壁に衝突するなど事故が乱発していたため、必ず機体を緊急停止できる人員を配置し、その人の許可<sup>\*3</sup>なく発進させてはいけないという管制塔システムを導入した。その結果安全性が向上し、機体を破壊することなく無事本番を迎えた。

---

<sup>\*3</sup> 電源を入れると発進する状態のときに“Cleared for run”の合図。後に電源を入れても自動で発進せず、管制官が発進信号を送るよう変更された。

# 第 5 章

## 評価

### 5.1 ループ速度

1 ループがどの程度の時間で回るか、というのはよい制御をするために非常に重要である。ボトルネックが大きかった当初、1 ループは 820 ms 程度であった。最終的には 1 ループが 203  $\mu$ s 程度となり、10 倍以上の高速化が達成された。

### 5.2 実測記録

530RPM のモータを使用した機体では 59 秒、1000RPM のモータを使用した機体では 65 秒であった。同じ速度ではローギアのモータの方が安定すること、およびチューニングにかけた時間が前者の方が長いことを考えれば妥当な結果である。

### 5.3 感想

この実験はライントレーサをフルスクラッチで製作し、爆速で走行させるという高い目標をもって始まりました。小中学生の頃に電子工作をかじっていて、基本的な回路の理解があり、プログラムも普通に書ける私（と望月氏）にとっては既製品の機体を少し改造して走行させるというはある意味当たり前にできることだ、という前提があったからです。無論、昨年まではラズパイと ev3 を用いるという制約があったので今年もそうであればここまで野性的な実験にはならなかったと思います。第一回目の授業で先生が、昨年と違ってすべて部品について自由で、何が必要か（そして、何が学びになるか）を考えて始めてほしい、という旨と情報共有は自由という旨が説明されたときから、望月氏とタッグを組んでこのプロジェクトは始まりました。

両者ともにソフトウェア、ハードウェアの両方ともある程度扱える素養はあれど、私はどちらかというとソフトウェア寄りの研究をこれまでにしていて、数学などを活用して理論的に考えていくことが得意なのに対し、望月氏はハードウェアの研究をしていましたから、ハードウェア側の設計は望月氏の方が長けていました。それゆえ得意分野が相補的な関係になっていたことはチームを組むうえで非常にメリットだったかと思います。私達が春学期に取っていた実験はどちらも複数人で進めるものでしたので、複数人で作業するのは実験の目的のひとつだろう、と思っていたし、オンライン世代というのもあるのですが、10 人もいるのにお通夜のような雰囲気の実験教室を非常に心配していましたから積極的にコミュニケーションが図れるように画策していました。ほぼ全員参加の discord を作った（これはあまり活用されなかった）、手が止まっている人や机

がきれいな人に声をかけて困りごとがないか聞いて回ったり（これはかなり有効で、いろいろ出てくる。ハードウェアの知識のない人が自由にしろ、壊すなと言われると何を調べるべきかもわからないというのはあたりまえ。質問をしに行くのも最初のうちは何を質問していいかわからないので難しい。）しました。効果の有無はわかりませんが、最終的には割と会話も生まれ、全員完成できたことは素晴らしいと思います。

少し話題が逸れましたが、このような背景があるため、私がこの実験を通じて学ぼうとしたことは、もちろんハードウェア寄りです。具体的には、知識として、大学レベルの電気回路の知識、制御工学の基礎について。そして、実践として、回路の設計やこれまで経験のないSMDのはんだ付け、カシメ工具でPHコネクタを通すなどの作業などです。

この中でも電気回路の知識は非常に身につけられたと思うので、紙面を割いて電気回路の解説を書いてみました。理解をしても自分で解説を書くのは存外難しく、導入があまり美しくないほか、計算を省略しがちなので対象が初心者とも上級者とも言い難い微妙な仕上がりになってしまったこと、書こうと計画していた部分まで書き上がらなかったのは反省点です。制御工学も軽くは勉強したのですが、解説できるほどではなかったことと時間が足りなかつたので解説は書きませんでした。

これらの知識を前提として、機体の設計はできています。私のスキル的にはハードウェアよりもソフトウェアのコントリビューションが多いですから、このレポートではハードウェアの部分は比較的少ないページ、ソフトウェアの部分は重点的に解説しています。

次に、本体に関する反省点なのですが、レポートを書くためにソースコードを整理してたらいくつか大きなバグが見つかりました。1分切りというかなりの好成績でコースを走破したあの機体なのですが、実はまだ本領発揮していなかった可能性があるようです。走行会後にこれが見つかったことだけが惜しい点でした。ほかには、元の設計どおりではうまく動かず、設計が度々変更されながらユニバーサル基板が増えていったことも反省点ではあるのですが、これは初めてのロボット製作であることを考えると致し方ないかと思います。

最後に、私事でございますが、このレポートは大学院飛び入学予定の私にとって学部最後のレポートになります。そのため、記念としてかなり本気を出して書きました。非常に長いことで先生に手間を取らせて申し訳ありませんでした。ここまでお読み頂きありがとうございました。

# 参考文献

- [1] 友男謙倉. 電気回路 (電子工学初步シリーズ 3・4). 培風館, 単行本, 9 1998.
- [2] トライイット. 電流・電圧とは ~電流・電圧のちがい、a(アンペア)と v(ボルト)~. [https://www.try-it.jp/keyword\\_articles/52/](https://www.try-it.jp/keyword_articles/52/).
- [3] 亀山寛. 電気回路と水流モデルとの類推に関する考察 (1) . <https://shizuoka.repo.nii.ac.jp/record/2336/files/090305003.pdf>.
- [4] いもあらい。Et ロボコンの要素技術。, 2011. <http://imoarai.cocolog-nifty.com/blog/2011/11/et-49b4.html>.

## 付録 A

# 設計の変遷

本付録では、最終成果物に至るまでの設計の変遷を示す。

### A.1 零号機

すべて ev3 を用いて授業初日に作り上げた機体。ev3 の性能評価のために使われた。この機体の評価により、初日にして ev3 を一切使わない設計に移行することを決定した<sup>\*1</sup>。

### A.2 初号機

10月中に作り上げた実験機。図 A.1 のように、配線はブレッドボードとジャンパ線、固定はマスキングテープ、土台にいくつかのレゴパーツ、センサは本番機同様のもの、モータはエンコーダなしギアなしの DC モータを直接搭載したプロトタイプである。加速力は非常に低いが、フィードバック制御がないため直線で際限なく速度が上がって次のカーブを曲がりきれずコースアウトする。一方でカーブが連続する地形では普通に走ることができた。回路設計はこの機体のときにはほぼ確定した。問題点を洗い出し、設計の原型を作ったという点で非常に重要な役割を果たしたプロトタイプである。これ以降は徐々に本番機に近づいていくが、変化が追いややすい部分ごと紹介する。

### A.3 ベースの設計

#### A.3.1 Version 1

初期には本年のコースはなかったので、昨年のコースに合わせた設計(図 A.2)。結論から言うと、今年のコースは遙かに去年のものより難しく、物理的に曲がり切ることができない設計であったためボツになった。この設計は高速走行を目指してホイールベースが大きめ<sup>\*2</sup>に作られていたほか、横幅もゆとりがあった。高速で走る走行体はセンサが車輪から遠いほど必要な回転量が減るので有利であることでこのような設計となっていたが、今年のコースの急なカーブに対応できず、改善が求められた。

---

<sup>\*1</sup>もちろん、そのほうが勉強になるという事実が後押しした。

<sup>\*2</sup>前輪がないのでこの用法は厳密には誤りなのかもしれないが、ここではセンサ下の接地部分と後輪間の距離をホイールベースと書いている。

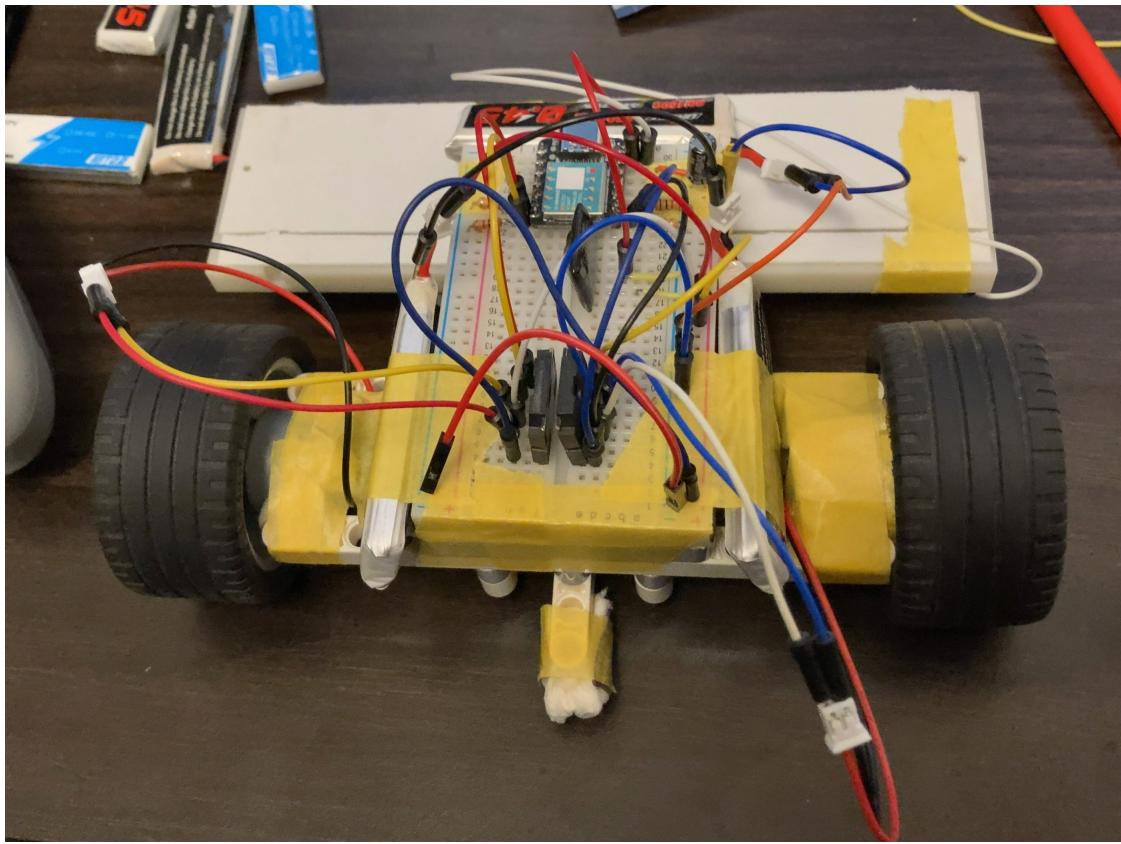


図 A.1 初号機

### A.3.2 Version 2

最終的に採用された設計。本編で解説した通りのものである。ホイールベース、横幅ともに限界まで縮め、旋回性能を上げた。

## A.4 メイン基板

### A.4.1 Version 1

最初に発注した基板。ピンアサインに問題があったので作り直しになった。

### A.4.2 Version 2

問題を修正し、ついでにパーツの配置を変えることで小型化した。

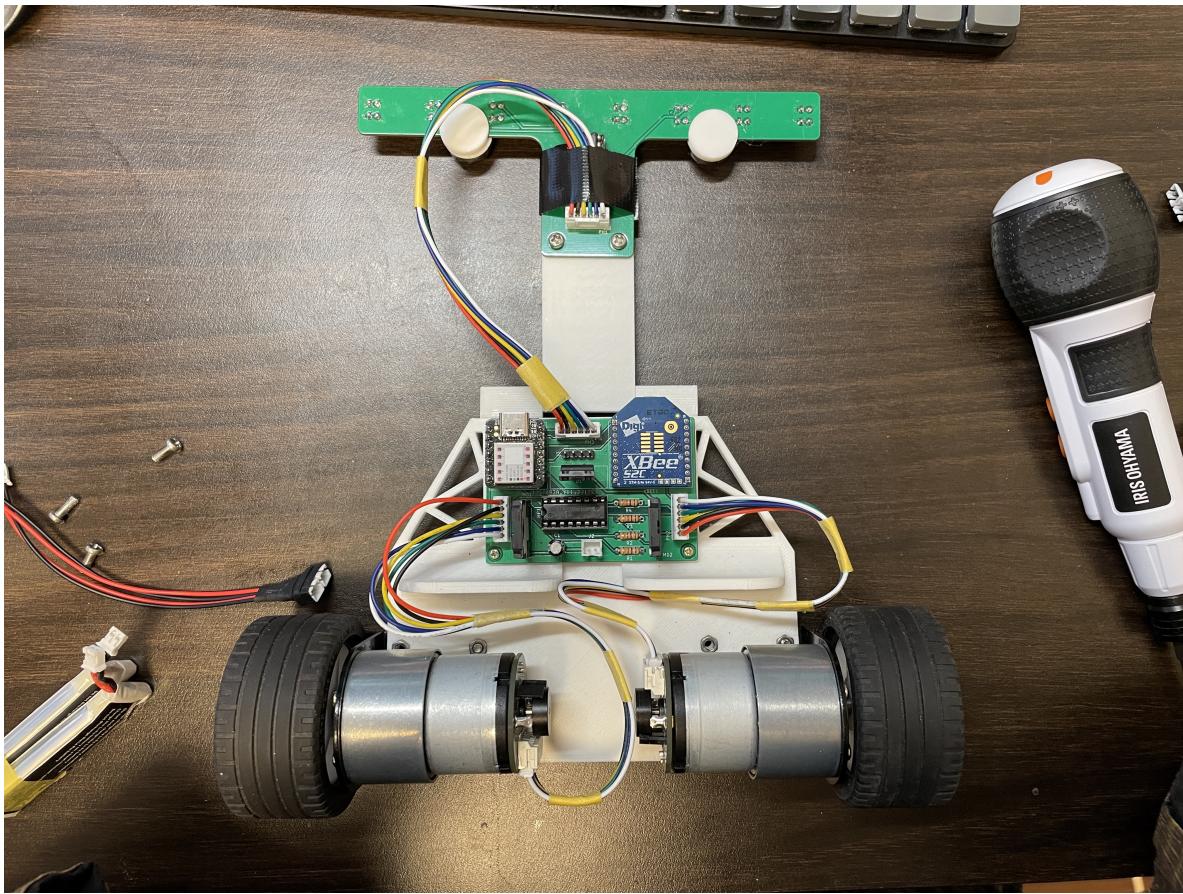


図 A.2 ベース Version 1 を採用した機体

#### A.4.3 Version 3

プリント基板自体が変更されたわけではないが、マルチプレクサを使わない設計に変更する必要が出てきたためユニバーサル基板で設計を少し拡張した。

### A.5 センサ値の統合

#### A.5.1 Version 1

各センサ  $i$  に単純な重み  $w(i) = -4 + i(0 \leq i \leq 3), -3 + i(4 \leq i \leq 7)$  を乗算し、単純に総和をとることで統合した。原因是この時点でははっきりしていなかったが、コースアウトすることが多かったため、改善に乗り出した。

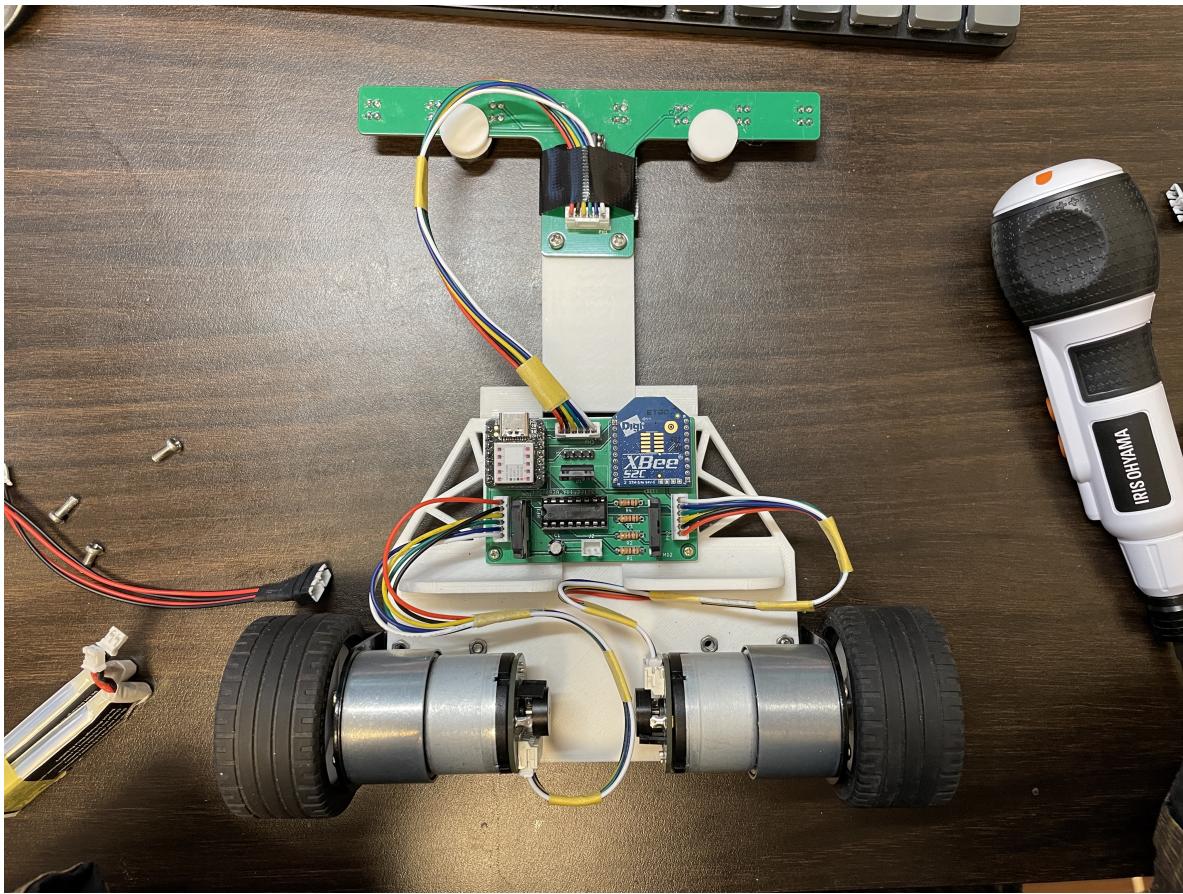


図 A.3 ベース Version 2 を採用した機体

#### A.5.2 Version 2

各センサの重みを車軸から見込んだ角度に変更した。センサ中心からの距離と  $\arcsin$  を用いた実装であった。多少改善が見られたが、振動が収まらなかった。

#### A.5.3 Version 3

最終的に採用されたシステム。重み付けして総和を出すのではなく、コース線の中心を推定することで単調性を確保し、安定したターンを実現した。角度を求める関数も三平方の定理が必要ない  $\arctan$  に変更され、計算量が減少した。

## 付録 B

# インシデント、問題解決集

### B.1 概要

ロボット製作をしていると、原因不明の問題が次々と発生し、時間がどんどん吸い取られていくことがよくある。本章では、開発中に実際にあったハマりどころを列挙し、原因を述べる。

### B.2 レギュレータの周波数特性

当初使用していた DC-DC レギュレータは高級な作りになっていて、レギュレータという名前にもかかわらず、コンバーダのような仕組みだった。その結果、モーターが回転し始めた瞬間の過渡現象的な電源の電圧降下を拾ってしまい、5V 出力のレギュレータが瞬間に 2V 程度になりマイコンが落ちてしまうという問題が発生した。電源側は 12V 程度供給していて、電圧降下したときも 9V 程度を維持していたからレギュレータ側に問題があると気づくのに遅れた。コンデンサで安定化させ、レギュレータをシンプルな 3 端子レギュレータに変更することで解決した。

### B.3 PH コネクタ接触不良

よくある話から。本機では、PH コネクタを用いていくつかの配線を束ねていたが、1 本だけ接触が悪いという状況は案外気づきにくく、「力を加えると精度が向上する」などと呼ばれていた。

### B.4 回路設計のミス

発注した基板の設計ミス。本来電源が供給されるべきところであるのに、浮島となっていて 0V となっているところがいくつかあった。基板表面を削って適当に電源が供給されているところとはんだ付けすることで解決した。

また、センサの基板で表裏を間違えていて、フォトセンサの素子の向きを全部直さないといけなくなった。

### B.5 マイコンのバグ

マイコンにプログラム書き込んで、そのまま使用すると 8 番ピンの出力電圧が不安定になるというバグ。書き込み後、一度電源と切り離し、再度電源と接続すると問題なく動作するため再現が難しく、ことあるごとに悩

まされた。個体差なのか仕様なのかは定かでない。

## B.6 温度特性

教室では良好に動いていたのに、家に持ち帰り確認すると全く違う値が出る。いったい何が違うのだろう？その答えは素子の温度特性。寒い外気にさらされて持ち帰ってきた機体は、抵抗器も半導体素子も別の特性を示すのである。現実の素子がいかに理想的でないか思い知らされる出来事である。

## B.7 何もしてないのに壊れた

一度も使用していなかったマルチプレクサが、正しく動作していなかった。初期不良か、静電気か。

## B.8 テスタの個体差

大きなプルアップ抵抗を使用していると、電圧を測ったとき、テスタ側の内部抵抗によって分圧され、思ったのと違う値になる。普通の回路では気にならないが、 $1\text{ M}\Omega$  の抵抗を使用していたため、テスタの内部抵抗と同じくらいのオーダになっていたようだ。安物のテスタでは 2.1 V を示し、もっと良いテスタでは 3.5 V ということが起こり、再現性の無さに苦労した。

## B.9 これらの組み合わせ

これらの問題が独立に発生しているなら、まだ苦労しなかったんだろう。しかし、2つまとめて起こる、ということが頻発したため非常に苦労した。

## 付録 C

### ソースコード

Listing C.1 main.ino

```
1 #define ML1 2
2 #define ML2 3
3 #define MR1 1
4 #define MR2 4
5 #define SENSOR_PIN 0
6 #define ENCODER_L 5
7 #define ENCODER_R 10
8 #define LED 13
9
10 #define RIGHT 0
11 #define LEFT 1
12
13 #define SENSOR_DISABLED false
14
15 float gearRatio;
16 float wheelDiameter;
17 float distancePerMotorRotate;
18
19 int ENCODER_PIN[2] = {ENCODER_R, ENCODER_L};
20
21 int isActive = true;
22
23 float currentSpeed[2] = {0,0};
24
25 //original map function which can map with float values
26 float mapFloat(float inputValue, float inputLower, float inputUpper, float outputLower,
27     float outputUpper){
28     float value = (outputUpper-outputLower)*(inputValue - inputLower) / (inputUpper -
29         inputLower) + outputLower;
30     return max(min(value, outputUpper), outputLower);
31 }
32
33 //motor power control function
34 void controlMotor(float Rspeed, float Lspeed){ //argument range -1 to 1
```

```

33     if(Rspeed>=0){
34         analogWrite(MR1, int(255*Rspeed));
35         analogWrite(MR2, 0);
36     }else{
37         analogWrite(MR2, int(255*(-Rspeed)));
38         analogWrite(MR1, 0);
39     }
40
41     if(Lspeed>=0){
42         analogWrite(ML1, int(255*Lspeed));
43         analogWrite(ML2, 0);
44     }else{
45         analogWrite(ML2, int(255*(-Lspeed)));
46         analogWrite(ML1, 0);
47     }
48 }
49
50 //chattering eliminator function for the binary hall sensors
51 int prevEncoderState[2] = {0,0};
52 int _smoothEncoderSignal(int encoderNum){
53     while(true){
54         int encoderSignals[3] = {0,0,0};
55         for(int i = 0; i < 3; i++){
56             encoderSignals[i] = digitalRead(ENCODER_PIN[encoderNum]);
57         }
58         if(encoderSignals[0]==encoderSignals[1] && encoderSignals[1]==encoderSignals[2]){
59             return encoderSignals[0];
60         }
61     }
62 }
63
64 //signal state detection function
65 bool _isChanged(int encoderNum){ //argument range 0 or 1
66     int encoderState = _smoothEncoderSignal(encoderNum);
67     bool changed = false;
68     if(prevEncoderState[encoderNum]==0 && encoderState==1){
69         changed = true;
70     }
71     prevEncoderState[encoderNum] = encoderState;
72     return changed;
73 }
74
75 //measure speed with parallel processing emulation (amazingly faster than before)
76 int deadLineTime = 50000;
77 int pulsePerRotate = 11;
78 int prevSignalTime[2] = {0,0};
79 float _getSpeedWithTime(int motorNum){
80     int nowTime = micros();
81     int deltaTime = nowTime - prevSignalTime[motorNum];

```

```

82     if(_isChanged(motorNum)){
83         float MotorFreq = 1000000/(deltaTime*pulsePerRotate);
84         float speed = MotorFreq*distancePerMotorRotate;//cm/s
85         prevSignalTime[motorNum] = nowTime;
86         return speed;
87     }else if(deltaTime > deadLineTime){
88         return 0;
89     }else{
90         return currentSpeed[motorNum];
91     }
92 }
93
94 //speed update function
95 void getSpeedWithTime(float *spd){
96     spd[RIGHT] = _getSpeedWithTime(RIGHT);
97     spd[LEFT] = _getSpeedWithTime(LEFT);
98 }
99
100 //smooth start speed function
101 void delayAccelerationSpeed(float targetSpeed, float *newTargetSpeed, float maxError =
102     10){
103     float speedDeltas[] = {targetSpeed-currentSpeed[RIGHT],
104                           targetSpeed-currentSpeed[LEFT]};
105     if(speedDeltas[0] >= maxError && speedDeltas[1] >= maxError){
106         newTargetSpeed[RIGHT] = min(currentSpeed[RIGHT] + maxError, targetSpeed);
107         newTargetSpeed[LEFT] = min(currentSpeed[LEFT] + maxError, targetSpeed);
108     }else{
109         newTargetSpeed[RIGHT] = targetSpeed;
110         newTargetSpeed[LEFT] = targetSpeed;
111     }
112     return;
113 }
114
115 float getLinePos(){
116     float linePos = mapFloat(float(analogRead(SENSOR_PIN)),7.0f,673.0f,-10.0f,10.0f);
117     return linePos;
118 }
119
120 float sensorErrorPrev;
121 float sensorErrorSum;
122 float getPIDsensorValue(int linePos){
123     float skP = 10.0;
124     float skI = 0.0002;
125     float skD = 12.5;
126
127     //on start
128     if(currentSpeed[RIGHT]<=30 && currentSpeed[LEFT]<=30){

```

```

129     skD = 8.0;
130
131 }
132 float error = linePos;
133 sensorErrorSum += error;
134 float PIDvalue = error*skP + sensorErrorSum*skI + (error - sensorErrorPrev)*skD;
135 sensorErrorPrev = error;
136 return PIDvalue;
137 }
138
139
140 float spdPrev[2];
141 float spdErrorPrev[2];
142 float spdErrorSum[2];
143 float power[2];
144 void controlEachMotorWithPID(float rightTargetSpeed, float leftTargetSpeed){
145     float mkP = 0.04;
146     float mkI = 0.000015;
147     float mkD = 0.05;
148
149     float spdError[2];
150
151     spdError[RIGHT] = rightTargetSpeed - currentSpeed[RIGHT];
152     spdError[LEFT] = leftTargetSpeed - currentSpeed[LEFT];
153     spdErrorSum[RIGHT] += spdError[RIGHT];
154     spdErrorSum[LEFT] += spdError[LEFT];
155     float pwrR = spdError[RIGHT]*mkP + (spdError[RIGHT]-spdErrorPrev[RIGHT])*mkD +
156         spdErrorSum[RIGHT]*mkI;
157     float pwrL = spdError[LEFT]*mkP + (spdError[LEFT]-spdErrorPrev[LEFT])*mkD +
158         spdErrorSum[LEFT]*mkI;
159     spdErrorPrev[RIGHT] = spdError[RIGHT];
160     spdErrorPrev[LEFT] = spdError[LEFT];
161     controlMotor(pwrR, pwrL);
162 }
163
164 float limitSpeedOnCorner(float targetSpeed, float deacceleratedSpeed, int linePositon){
165     float spdDelta = (targetSpeed - deacceleratedSpeed)/6;
166     float spd = targetSpeed - spdDelta*abs(float(linePositon));
167     return spd;
168 }
169
170 void controlMotorFromSensors(float targetSpeed, float deacceleratedSpeed=10){
171     int linePos = getLinePos();
172     float sensorPIDValue = getPIDsensorValue(linePos);
173     float newTargetSpeed[2];
174     delayAccelerationSpeed(targetSpeed, newTargetSpeed, deacceleratedSpeed);
175     float rightSpeed = newTargetSpeed[RIGHT] - sensorPIDValue;
176     float leftSpeed = newTargetSpeed[LEFT] + sensorPIDValue;
177     controlEachMotorWithPID(rightSpeed, leftSpeed);

```

```

176 }
177
178 void showSpeed(){
179     Serial1.print("Speed: ");
180     Serial1.print(currentSpeed[RIGHT]);
181     Serial1.print(", ");
182     Serial1.print(80);
183     Serial1.print(", ");
184     Serial1.println(20);
185 }
186
187 void showLinePos(){
188     Serial.println(getLinePos());
189 }
190
191 int initialMicros;
192 int loopCount = 0;
193 void showLoopSpeed(int loopNum){
194     if(loopCount==0){
195         initialMicros = micros();
196     }
197     loopCount++;
198
199     if(loopCount==loopNum){
200         Serial.print("Loop Speed ( ");
201         Serial.print(loopNum);
202         Serial.print(" iters): ");
203         Serial.print((micros()-initialMicros)/loopNum);
204         Serial.println("us");
205         loopCount = 0;
206         showSpeed();
207     }
208 }
209
210
211 void initAll(){
212     //global variables initialize
213     currentSpeed[RIGHT] = 0;
214     currentSpeed[LEFT] = 0;
215     sensorErrorPrev = 0;
216     sensorErrorSum = 0;
217     spdErrorPrev[RIGHT] = 0;
218     spdErrorPrev[LEFT] = 0;
219     spdErrorSum[RIGHT] = 0;
220     spdErrorSum[LEFT] = 0;
221     initialMicros = 0;
222     loopCount = 0;
223
224     //Motors init

```

```

225     pinMode(MR1, OUTPUT);
226     pinMode(MR2, OUTPUT);
227     pinMode(ML1, OUTPUT);
228     pinMode(ML2, OUTPUT);
229     analogWrite(MR1, 0);
230     analogWrite(MR2, 0);
231     analogWrite(ML1, 0);
232     analogWrite(ML2, 0);
233
234     //encoder init
235     pinMode(ENCODER_L, INPUT);
236     pinMode(ENCODER_R, INPUT);
237
238     //multiplexer input pins init
239     pinMode(SENSOR_PIN, INPUT);
240
241     //set the hardware
242     gearRatio = 18.8;
243     wheelDiameter = 5.6;
244     distancePerMotorRotate = wheelDiameter*PI/gearRatio;
245 }
246
247 void Xbee(){
248     if(Serial1.available()){
249         char c = Serial1.read();
250         switch (c){
251             case 's':
252                 initAll();
253                 isActive = false;
254                 break;
255
256             case 'b':
257                 initAll();
258                 isActive = true;
259                 break;
260             default: break;
261         }
262     }
263 }
264
265 void setup() {
266     Serial.begin(9600);
267     Serial1.begin(9600);
268     initAll();
269 }
270
271 void loop() {
272     Xbee();
273     if (isActive){

```

```

274     getSpeedWithTime(currentSpeed);
275     controlMotorFromSensors(100);
276 }else{
277     controlEachMotorWithPID(0,0);
278 }
279 }
```

Listing C.2 sensor.ino

```

1 #define LED 13
2 #define SIGNAL_OUTPIN 0
3
4 int sensorPins[] = {3,9,10,2,1,4,6,5};
5 int sensorMaxValue = 1023;
6 int sensorMinValue = 0;
7 int linePos.MaxValue = 10;
8 int linePos.MinValue = -10;
9 float arctan7_13;
10
11 float mapFloat(float inputValue, float inputLower, float inputUpper, float outputLower,
12                 float outputUpper){
12     float value = (outputUpper-outputLower)*(inputValue - inputLower) / (inputUpper -
13                           inputLower) + outputLower;
13     return max(min(value, outputUpper), outputLower);
14 }
15
16 float ReadSensorBinary(int sensorNum){
17     int state = 0;
18     if( analogRead(sensorPins[sensorNum]) >1020) state = 1;
19     return state;
20 }
21
22 float ReadSensor(int sensorNum){
23     return mapFloat(analogRead(sensorPins[sensorNum]),0,1023,0,1);
24 }
25
26 int getLinePos(){
27     int linePos = 0;
28     for(int i=0; i<8; i++){
29         int weight = i-3;
30         if(weight<=0) weight--;
31         linePos += ReadSensorBinary(i)*weight;
32     }
33     return linePos;
34 }
35
36
37 float thetaWeight[] = {-6.45, -4.79, -2.95, -1.0, 1.0, 2.95, 4.79, 6.45};
38 float getLinePosThetaWeighted(){
39     float linePos = 0;
```

```

40     for(int i=0; i<8; i++){
41         linePos += ReadSensorBinary(i)*thetaWeight[i];
42     }
43     return linePos;
44 }
45
46 float sensorDistance[] = {-7, -5, -3, -1, 1, 3, 5, 7};
47 float getThetaWithArctan(){
48     float theta = 0;
49     float distance = 0;
50     int sensorBlackCount = 0;
51     for(int i=0; i<8; i++){
52         int sensorValue = ReadSensorBinary(i);
53         distance += sensorValue*sensorDistance[i];
54         sensorBlackCount += sensorValue;
55     }
56     if(sensorBlackCount>0){
57         distance = distance/sensorBlackCount;
58         theta = atan(distance/13);
59         // Serial.println(theta);
60         // Serial.println(sensorBlackCount);
61         return theta;
62     }else{
63         return 0;
64     }
65 }
66
67 void initAll(){
68     pinMode(LED, OUTPUT);
69     pinMode(SIGNAL_OUTPIN, OUTPUT);
70     pinMode(sensorPins[0], INPUT);
71     pinMode(sensorPins[1], INPUT);
72     pinMode(sensorPins[2], INPUT);
73     pinMode(sensorPins[3], INPUT);
74     pinMode(sensorPins[4], INPUT);
75     pinMode(sensorPins[5], INPUT);
76     pinMode(sensorPins[6], INPUT);
77     pinMode(sensorPins[7], INPUT);
78     arctan7_13 = atan(7.0/13.0);
79 }
80 }
81
82 void setup(){
83     initAll();
84 }
85
86 void loop(){
87     analogWrite(SIGNAL_OUTPIN,
88                 mapFloat(getThetaWithArctan(),-arctan7_13,arctan7_13,0,700));

```

```
88 //Serial.println(mapFloat(getThetaWithArctan(),-arctan7_13, arctan7_13,0,700));
89 // for(int i=0; i<8; i++){
90 //   Serial.print(analogRead(sensorPins[i]));
91 //   Serial.print(", ");
92 // }
93 // Serial.println();
94 // Serial.println(getThetaWithArctan());
95 }
```