

## 情報工学実験3

チームないん

17T1697W 大石 健

17T1616M 藤井 瞭

17T1679H 江原 仁美

17T1632X 田中 悠

17T1653A 澤田 耀介

2020年2月5日

# 目次

<b>第1章</b>	<b>プロジェクト管理</b>	<b>1</b>
1.1	一般的なプロジェクト管理の手法	1
1.1.1	プロジェクトとプロセスについて	1
1.1.2	プロジェクトの進め方	1
1.1.3	実際の遂行	1
1.1.4	時間管理とコスト管理の手法	2
1.2	開発モデルの検討	2
1.2.1	ウォーターフォールモデル	2
1.2.2	プロトタイピングモデル	3
1.2.3	アジャイル開発	3
1.3	プロジェクト全体像の明確化	3
1.3.1	要求の把握	3
1.3.2	プロジェクト目標の文書化	4
1.3.3	予期されるプロジェクトの最終成果物	4
1.4	作業の分解・詳細化	4
1.5	役割分担と所要期間の見積り	5
1.6	ネットワーク図の作成	5
1.7	スケジュールの構築	5
1.8	負荷の調整	6
1.8.1	結果	6
1.9	進捗の管理	7
<b>第2章</b>	<b>ライントレーサ</b>	<b>9</b>
2.1	ライントレースについて	9
2.1.1	ライントレース大会の概要	9
2.2	ハードウェアの構成	9
2.2.1	ハード全体について	9
2.2.2	ハードの工夫点	11
2.2.3	センサ回路	11
2.2.4	モータドライバ	11
2.2.5	ログ送信の手法	12
2.3	ロボットの構成	13
2.4	ソフトウェア	15
2.4.1	制御方法	15
2.4.2	基本アルゴリズム	16

2.4.3	中級用ソースコード	18
2.4.4	上級用ソースコード	18
2.4.5	PID 制御	19
2.4.6	on-off スイッチ	21
2.5	日々の記録	24
2.5.1	10 月 3 日	24
2.5.2	10 月 10 日	25
2.5.3	10 月 17 日	26
2.5.4	10 月 24 日	27
2.5.5	11 月 7 日	28
2.5.6	11 月 14 日	29
2.5.7	11 月 21 日	31
2.5.8	11 月 28 日	32
2.6	大会の考察	33
2.6.1	プレ大会	33
2.6.2	本大会	33
2.6.3	追従大会に向けて	33
2.7	今後の展望	34
2.7.1	センサの改良	34
2.7.2	定数調節の実装	35
2.7.3	ラインセンサの配置の検討	38
2.7.4	バッテリー電圧の読み出し	38
2.7.5	センサの電源の安定化	39
2.7.6	基板の使用	39
2.8	ライントレーサのまとめ	39
<b>第 3 章</b>	<b>追従ロボット</b>	<b>49</b>
3.1	追跡ロボットとは	49
3.1.1	追跡ロボットについて	49
3.1.2	大会概要	49
3.2	ハードウェアの構成	49
3.2.1	回路図	49
3.3	メカニカル	50
3.3.1	機体の構成	50
3.4	超音波センサについて	51
3.4.1	超音波センサーについて	51
3.5	ソフトウェア	54
3.5.1	ソフトウェアの構成	54
3.5.2	対象の検出	55
3.5.3	2 輪差動駆動	56
3.5.4	混線の除去	56
3.6	先導機体	57

3.7	日誌 . . . . .	58
3.7.1	12月5日 . . . . .	58
3.7.2	12月12日 . . . . .	60
3.7.3	12月19日 . . . . .	61
3.7.4	12月26日 . . . . .	62
3.7.5	1月9日 . . . . .	63
3.7.6	1月16日 . . . . .	64
3.8	追従大会 . . . . .	65
3.8.1	プレ大会考察 . . . . .	65
3.8.2	本大会結果 . . . . .	65
3.8.3	本大会考察 . . . . .	66
3.8.4	追従大会のまとめ . . . . .	66
3.9	今後の展望 . . . . .	66
3.9.1	超音波センサの追加 . . . . .	66
3.9.2	SLAMの実装 . . . . .	66
3.10	まとめ . . . . .	66

# 第1章 プロジェクト管理

この章では、少人数のプロジェクトを円滑に進めるために本チームが行った工夫について記述する。

## 1.1 一般的なプロジェクト管理の手法

プロジェクト管理を実際に行うに当たって、本実験および計算化学 IV の講義資料を参考にした。このセクションでは、一般に行われているプロジェクト管理の手法や開発手順の技術について述べ、それを実験にどう活用できるかを考察する。

### 1.1.1 プロジェクトとプロセスについて

プロジェクトとは、通常の業務に収まらない目標を達成するために、期間を限定して行う一連の作業のことをいう。本実験では、通常の授業では体験出来ないスケジュール管理やチームの役割分担、ハードウェアに合わせたソフトウェアの開発を行うことが大きな目標である。期間は 10/3~1/16 の約 3ヶ月間に及び、実験が終了した後はチームの最終報告書を提出する。

一方でプロセスとは、プロジェクトにおいて実行される一連の作業を遂行する手順のことである。本実験では、多くのメンバーが Arduino を用いたロボット開発をするのが初めてだと予想される。このような状況では、最終成果物に応じて必要な作業を事前に定められるとは限らないため、後述するアジャイル開発のようなチーム内でのコミュニケーションを重視するプロセスを辿ることが必要だと考えられる。

### 1.1.2 プロジェクトの進め方

プロジェクトを立ち上げた後は計画と実行を交互に繰り返して終結へと向かう。その間には適切な監視・コントロールが働き、計画の妥当性や成果物の確認が入る。本実験では必ず開始時に作業計画を立ててから作業を実行していたので、計画と実行のサイクルを厳密に守ってプロジェクトを遂行したと言える。また、実験における監視役は TA であり、毎回の授業前後に計画書または報告書を確認することでプロジェクトの破綻を防ぐ。

### 1.1.3 実際の遂行

3ヶ月に渡るプロジェクトの中で誤った計画を立てないために、まずプロジェクト全体像の明確化を行った。このフェーズでは要求の把握や要求の仕様化をし、プロジェクト終

了時の成果物を明確にする。

次に、作業の分解と詳細化を行う。一般的な手法としてはWBS<sup>1</sup>がある。WSBはまず抽象度の高い作業をトップに置き、それらを細かく分解して下っていき、末端に実作業を配置する木構造である。そして各ワークパッケージに責任者と成果物を決めて、責任者が常にその作業の進捗状況を把握するように義務付ける。実際の現場では作業の分解と詳細化をあまり行わなかった。理由は、授業開始時に抽象度の高い目標を立て、それを達成するための作業を箇条書きでまとめて計画したためである。よってWBSは使用せず、実作業のみを作業分担表に記録した。作業計画の際には1つの作業に責任者だけでなく支援者を割り当てることで負荷の分散を試みた。

#### 1.1.4 時間管理とコスト管理の手法

時間管理は、定期的に各作業の進捗を可能な限り定量的に評価することが求められる。特に自己申告での進捗報告に依存していると、実際の進捗と記録との間に大幅な差異が生じる可能性があるため、成果物を元に判断する習慣付けが必要である。

また、ビジネスにおけるプロジェクト管理では時間管理とコスト管理の両方を要求されるため、工数なるべく正確に見積もり、各工程への配分を決めて実績管理と完成時の予測を行う必要がある。代表的な見積もりの手法には経験に基づく類推見積もりや、対象としているプログラムの機能の数を計測することで工数を見積もるファンクションポイント法、プログラムの作業量とコストの関係を計測するモデルである Constructive cost model、期間とコストを連動させて管理するためのアードバリュー管理がある。今回の実験では類推見積もりを行なった。

## 1.2 開発モデルの検討

このセクションでは、ソフトウェア開発における一般的なライフサイクルモデルとプロセスの手法について述べ、本実験のソフトウェア開発で最も適している手法を検討する。

### 1.2.1 ウォーターフォールモデル

ウォーターフォールモデルは最も古いライフサイクルモデルであり、過去の大規模システムはこのモデルに基づいて開発が行われていた。このモデルではフェーズを明確に区切り、下のフェーズから上のフェーズへの手戻りを許さない。そのため非常に工程の進捗管理がしやすくなるのが利点だが、テストフェーズに到達するまで動作確認出来ないという欠点を抱えている。

ライントレースロボットの開発については、このモデルを参考にするのは適当でないと判断した。チームメンバー全員が未経験だったため仕様を固めることが出来ないからである。また、もし仕様を決められたとしても大会のルールやコースに変更があった場合に修正が出来なくなってしまう。開発の途中で要件が変更されるリスクを抱えることは現実的でないため、他のモデルを検討する必要がある。

---

<sup>1</sup>Work Breakdown Structure の略

## 1.2.2 プロトタイピングモデル

プロトタイピングモデルでは、開発の初期段階においてプロトタイプを短期間で作成、評価し、改善点を再び要求仕様に反映させてサイクルを繰り返す。満足のいく評価を得られた段階で、その時の仕様を確定して実用システムを作成する。プロトタイプモデルの利点は、過去に開発経験のない新規機能を含むソフトウェアシステムを開発する時に、ユーザーの曖昧な要求を仕様に落とし込めることである。

本実験では、実際に動作するシステムをなるべく早くハードウェアに搭載して仕様の妥当性を検証することで大会に間に合わないリスクを回避できる。よってプロトタイピングモデルとの相性が良いと考えられる。ただし、設計を詰めずにプログラムを作成してデバッグしながら修正している作業形態になると、無計画な開発に陥りソフトウェアの構造も複雑になってしまうので、プロセスを管理することが重要である。

## 1.2.3 アジャイル開発

アジャイル開発は迅速かつ適応的にソフトウェア開発を行う開発手法の総称である。アジャイルソフトウェア開発宣言では、プロセスやツールよりも個人と対話を、包括的なドキュメントよりも動くソフトウェアを、計画に従うことよりも変化への対応を価値とすることが述べられている。ライントレース大会および追従大会では上級コース完走時の配点が高く、大会開始時点で確実に上級コースを完走できる機体を用意することが重要だった。そのため、実働可能なソフトウェアの納品を頻繁に行うことを原則とするアジャイル開発の考え方を応用することで、大会で失敗するリスクを最小限に抑えることが出来た。

ソフトウェア開発については、コーディングするメンバーを藤井、田中、大石の3人に絞り、そのメンバー同士でソフトウェアの仕様や設計といった情報を直接話し合っ共有した。最もソフトウェア開発の経験豊富だった大石が積極的に他のメンバーと連絡をし、チャットツール選択や、バージョン管理システムの使い方を教えて開発環境を整えた。これによりアジャイル開発で重視されている個人との対話を円滑に行うことが可能となり、個人の進捗がチーム内で頻繁に共有されるため、プロトタイピングモデルにおける無計画な開発のリスクを抑えることが出来た。ソフトウェアの分量が増えても、ソフトウェアの開発メンバー全員がコードの中身を把握している上に、長期間の保守が必要ではなかったので大きな問題にはならなかった。

また、コーディングに慣れていないメンバーのキャッチアップのためにペアプログラミングを頻繁に行った。大石が持っている知識や技術を他のメンバーに共有することで、藤井と田中がソフトウェア作成の作業を単独で行えるようになった。

## 1.3 プロジェクト全体像の明確化

### 1.3.1 要求の把握

このプロジェクトで要求されているのは以下の3点である。

- プロジェクト型の活動を通じて、役割分担、スケジュール管理などを実体験する

要求	判断基準
役割分担, スケジュール管理を体験する	班員ごとの負荷を均等に割り振ったか. 実作業とズレなくスケジューリング出来たか.
実機上でソフトウェアを動作させる.	ハードに合わせてプログラムが記述されているか. XBeeの通信で適切にボーレートの値を調整したか.
効果的なプレゼン技術を身につける.	他のチームから高い評価を得られたか. チームが持つ知見を他のチームにも共有できたか.

表 1.1: プロジェクト目標

- 実機上で動作させることで, ソフトウェアが必ずしも思い通りに動かないことを見にしみて実感する.
- 効果的なプレゼン技術を身につける.

したがって, ライントレース大会で良い結果を残すだけではこのプロジェクトを成功させたという判断は出来ない. チームでの活動を成功させるためには, 各実験でチームメンバーをタスクに適切に配置し効率よく進捗を生み出すことが必要である. また, プレゼン大会では資料作成と発表練習の両方に時間を要するので, 準備期間を適切に設定して作業をすることが必要だ. 具体的には資料作成を手の空いたメンバーに役割分担をすることや, プレゼン大会の2週間前には準備を開始することなどが挙げられる.

### 1.3.2 プロジェクト目標の文書化

このプロジェクトの成功基準は, プロジェクトの要求を達成できたかどうかである. よって, プロジェクト要求とそれに対する成功基準を定め表 1.1 にまとめた.

### 1.3.3 予期されるプロジェクトの最終成果物

プロジェクト目標から予測される成果物は, ライントレーサーのハードウェア及び制御プログラムと, 追従機体のハードウェア及び制御プログラムである. また, プロジェクト進行中に作成したガントチャートや, 実験 moodle にアップロードしたチーム成果報告書, チーム wiki も最終成果物に含まれる.

## 1.4 作業の分解・詳細化

作業の分解を毎回の実験の前に行なった. 大会の日が近付くにつれ, 作業の種類自体は減っていき詳細化することがなくなっていく. よって, 作業の分解が十分でないことにより問題が生じることはなかった.



## 1.5 役割分担と所要期間の見積り

役割を大まかにプロジェクト管理, ソフトウェア, ハードウェアの3つに分担した。ただし, 作業開始時に全ての作業を予測して分担することは出来なかったため, 紙の進捗報告書には当初予定されていた作業のみを記入し, スプレッドシートの作業分担表には後から追加された作業も含めて記入することにした。プロジェクト管理は藤井が担当し, スケジュールの設定やガントチャート・作業分担表の記入を行なった。ソフトウェア開発は澤田と田中が担当し, 機体制御プログラムの作成やバージョン管理を行なった。ハードウェア開発は江原と大石が担当した。次に, 作業分担表を示す。これには実験中に取り組んだ全ての作業とその担当者が記録されている。

作業分担表からわかることは, 10/24(第4回)までの作業においては作業を詳細化して割り当てられていたが, 機体が組み上がってくると作業の種類が減って行ったということである。また, 実験が進むにつれメンバーの得手不得手がわかってくるので, 作業の種類ごとにアサインされるメンバーが偏っていくのがわかる。例えばXBeeでのリモート通信には藤井と大石が, PID制御の値調整は田中と大石が多くアサインされていた。実際にプロジェクトを進めると並行して進めることの出来る作業が少なかったため, 手が空いたメンバーを実験レポート作成やチーム報告の記述といったロボット制作以外の作業にアサインすることで解決した。

## 1.6 ネットワーク図の作成

実験のプロセスではネットワーク図を作成しなかったが, プロジェクト管理の妥当性を判断するために, 実験終了後にネットワーク図を作成して考察を行った。

ライントレース大会までのワークパッケージ間の依存関係を示すネットワーク図と検出したクリティカルパスが図1.1である。クリティカルパスの中にあるXBeeで2台のArduinoを接続する作業は技術的に可能かどうか不明だったので, PID制御の調整開始が遅れる可能性があった。この作業は締め切りを決め, 技術的に不可能だった場合は早めに見切ることにした。

追従大会までのワークパッケージ間の依存関係を示すネットワーク図とクリティカルパスを図1.2に示す。クリティカルパス中にある先導機体の調整に大幅な遅延が生じたため, 追従機体の調整までに1日ほど遅れが生じてしまった。これは予測出来た事態だが, トラブルを回避するのは容易ではない。対応として, 追従機体の調整が出来ず手の空いたメンバーはレポート作成を進めることでなるべく作業時間を有効に使うようにした。

## 1.7 スケジュールの構築

スケジュール構築を図1.3(a)に示すようなガントチャートを作成して行った。各実験の終了時に次回の実験の予定を記入しておき, 予想時間と実際にかかった時間との差分を記録した。ライントレース大会までの作業では, リモートでPIDゲインの値を調整するプログラムの作成に大幅な遅れが出たが, 他の作業との依存関係が無かったため全体には遅延が生じなかった。しかし追従大会に向けての作業では, 先導機体の調整に大幅な遅延が出た。それにより, 先導機体の調整作業に依存している他の作業にも遅れが生じた。

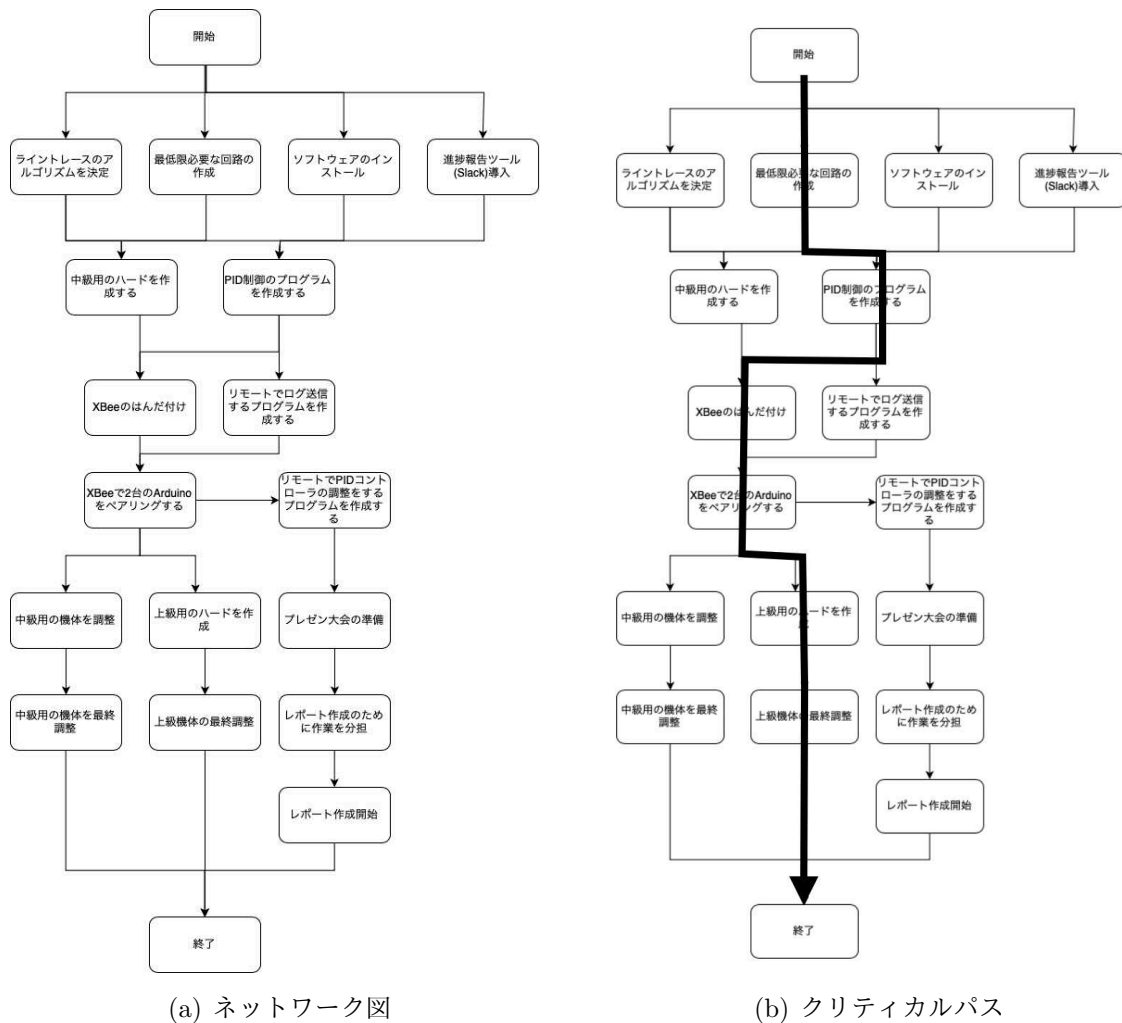


図 1.1: ライトレース大会までのタスクにおけるネットワーク図とクリティカルパス

## 1.8 負荷の調整

作業分担表を作成する時点で作業量の予測をし、メンバーに均等に割り当てられるようにして負荷を調整した。ハードウェアやソフトウェアの作業が少ない日はレポートやプレゼンテーションの資料作成をタスクに追加し、作業量が少ないメンバーに割り当てた。

### 1.8.1 結果

チーム内では、大石が最もハードとソフトの両方に詳しく、実際の作業量が最も多かった。また、授業時間外でも作業を行っていたので記録されている以上に作業量が多い。例えばリモートでtex ファイルのビルドを行うようにしたり、超音波センサーの特性を調べるためにラプラシアンフィルターを利用して波動のシミュレーションを行ったりした。これらの作業は必ずしも必要ではないが、他の作業を効率化したり、実験の内容を充実するために行われた。作業量の調整を行うために、機体の作成にあまり関わっていないメンバーにプレゼン資料やレポートの作成を多く割り当てた。

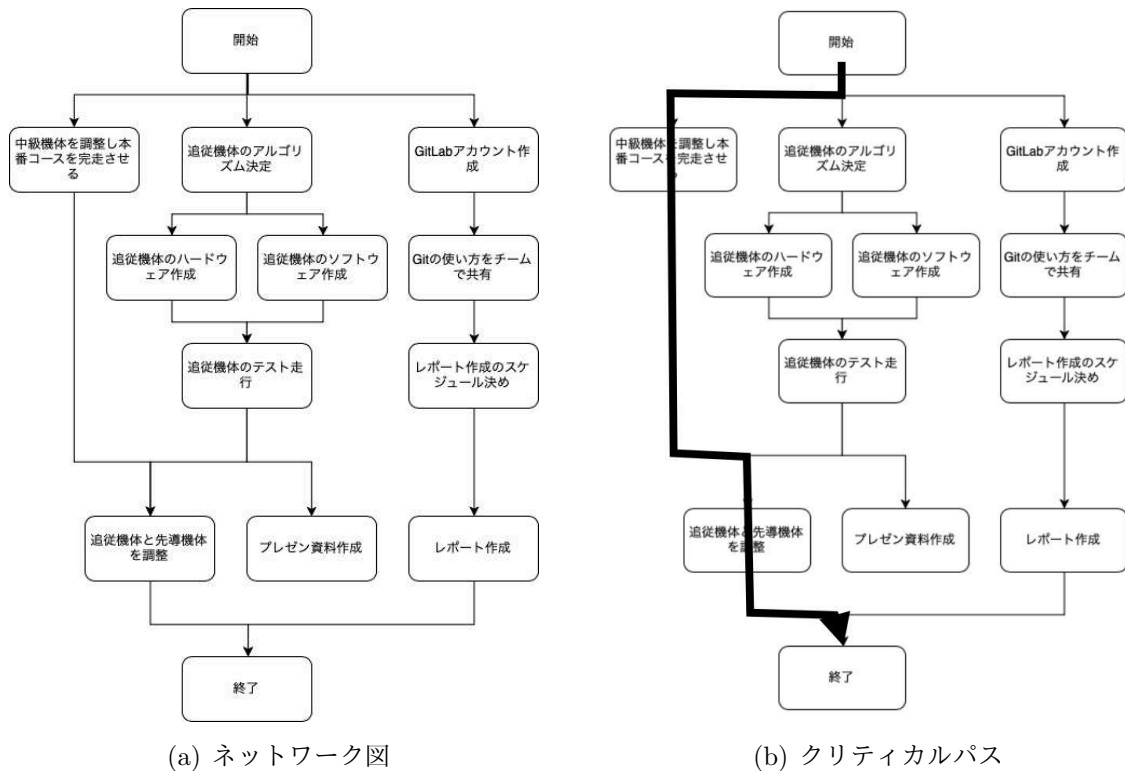


図 1.2: 追従大会までのタスクにおけるネットワーク図とクリティカルパス



(a) ライントレース大会までのガントチャート

## 1.9 進捗の管理

進捗を評価するために、各実験の最初で配布されるシートには詳細に達成度を記入した。そして、作業の遅れを把握するためにガントチャートに実績を埋めた。しかし、自己申告では作業の進捗を正確に把握することが難しいと判断したので、実験中はPMと大石が積極的にメンバーに声をかけて進捗を確認した。また、プログラムや制御方法の調査結果といった成果物を逐一チーム内 Slack にアップロードすることにした。これによりハードウェア

の担当メンバーもソフトウェア開発の状況や実際のプログラムを把握できるようになった。レポートの作業を分担するために Git を導入した後は、コミットの記録を遡ることにより進捗を把握しやすくなった<sup>2</sup>。

---

<sup>2</sup>編集長よりを参照

## 第2章 ライントレーサ

### 2.1 ライントレースについて

ライントレーサとは

ライントレーサとは、床に描かれたラインをなぞるように走り、1周あたりにかかる時間等を競うものである。ロボットの光学センサ部でラインの位置を読み取り、その値に応じて機体のステアリング制御を行う。

#### 2.1.1 ライントレース大会の概要

まず、ライントレース大会の概要について説明する。大会では、Arduinoを用いたライントレースを行う。初級、中級、上級の3つのコースが用意され、一周するタイムを競う。難易度が上がるほど、また、タイムが早いほど、得点が高い。配布されたArduino以外は使用してはならず、遠隔制御も禁止されている。同時に2台を走行させることも禁止されている。初級は直線のコースであり、中級は円周状のコースである。上級はくねくねした交差のないコースであり、このコースは曲がるところが多い。

中級、上級に関して以下の計算式を用いて得点 $p$ が与えられる。

$$p_{\text{中級}} = 40 \frac{t_1}{t} + 10$$

$$p_{\text{上級}} = 50 \frac{t_1}{t} + 50$$

### 2.2 ハードウェアの構成

#### 2.2.1 ハード全体について

全体の回路図を図2.1に示す。モジュールの回路図を図2.2に示す。なお、回路図はモジュール化されているため、同じ回路が複数あり、部品のリファレンス番号のみが異なるだけなので全ては載せていない。また、上級コース向けのロボットの回路図を示しているが、中級コース向けはセンサの数が2つに減らされている。

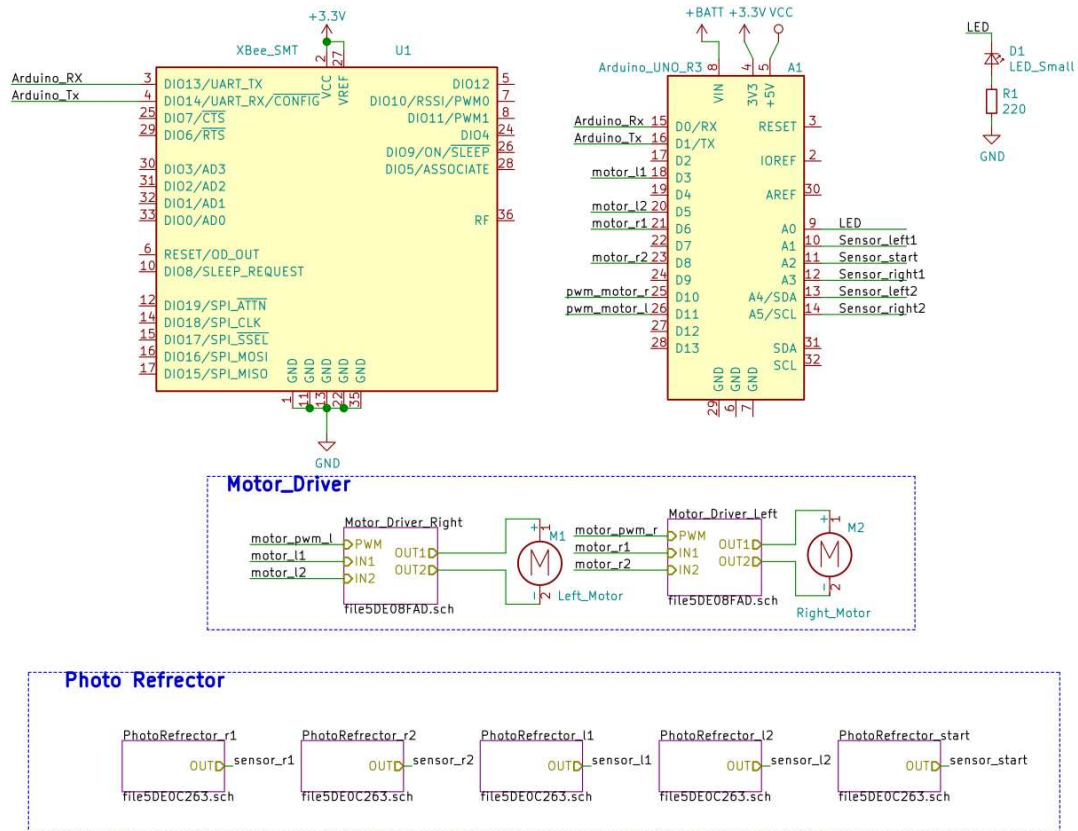


図 2.1: ライトレーサの全体回路図

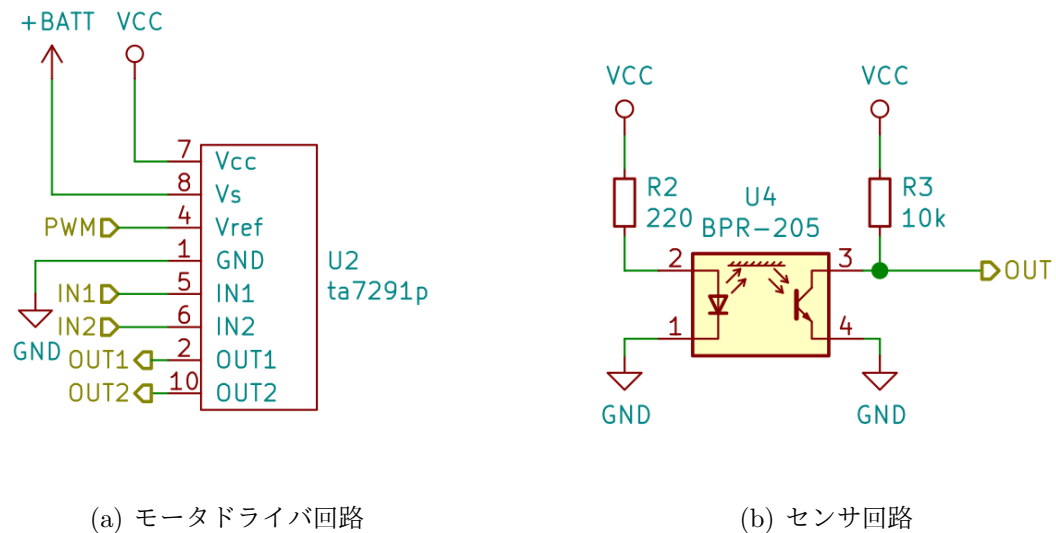


図 2.2: モジュールの回路図

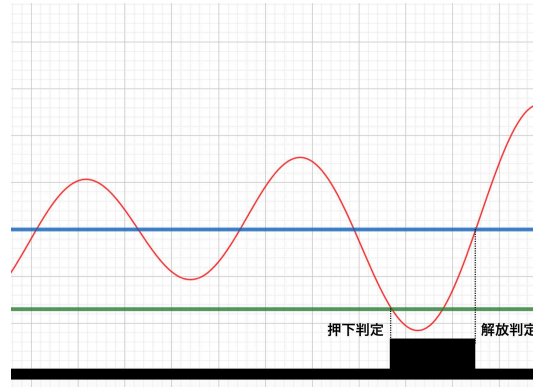


図 2.3: フォトリフレクタによる判定の図

## 2.2.2 ハードの工夫点

### 2.2.3 センサ回路

センサ回路が指示書のリファレンス回路とは異なる。R2のLEDの電流制限抵抗を  $1[k\Omega]$  のでは、製品仕様で想定されている光量に届かず、検出が不安定になるため  $220[\Omega]$  まで下げた。このときLEDに流れる電流は  $(5 - 1.4) \div 220 \approx 16.4[mA]$  となり、これでも定格内で動作することがわかる。また、可変抵抗は使用せず、入力インピーダンスは最大  $10[k\Omega]$  になるように、 $10[k\Omega]$  によるプルアップのみにした。

#### フォトリフレクタによるスタートボタンの代替

スタートスイッチを、フォトリフレクタで代用することにした。バッテリーのスライドスイッチは使いにくく、そのうえ書き込むと突然動いてしまう。押下判定と開放判定に2段階のレベルを設けて、判定を行う。フォトリフレクタの値を130まで下がった場合に押下判定を行い300を超えた時に開放判定を行った。図4において、押下の判定は緑のレベルで行い、ボタンを離れた判定は青のレベルで行う。これにより、安定してスタートとストップの判定を行うことができる。

スタートすると、LEDが点灯して知らせてくれる。LCDによる表示も考えたが、LCDではArduinoの空きポートが足りなかったなのでLEDによる表示のみとした。

### 2.2.4 モータドライバ

モータを駆動するモータドライバは、TA7291pを用いる。このモータドライバICは、中にリファレンス電圧を入力し、その電圧をモータの出力電圧へレギュレーションすることができる[1]。モータの電源は、バッテリー  $V_{BAT}$  から直接供給する。モータドライバIC内部のレギュレータを通して、安定した電圧を出力することができる。

モータの出力の電圧はリファレンス電圧から、少々オフセットがかかった状態で出力される。リファレンス電圧はArduinoからのPWM出力を平滑化して生成される。



図 2.4: UART のクロス接続

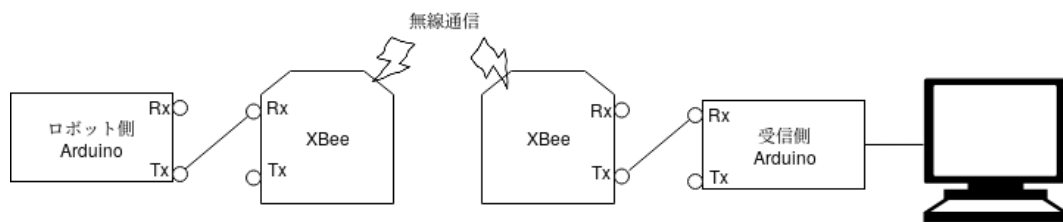


図 2.5: ペアリングの図

## 2.2.5 ログ送信の手法

### XBee によるデバックシステムの構築

XBee によるデバックシステムの構築 XBee は無線通信を実現し, 走行中にロボットと PC を間接的に繋ぐことで安全にログを取ることができる. Xbee は UART という規格を使って通信している. 図 2.4 に示すように, UART は Tx と Rx を繋ぐ, "クロス接続" にする. XBee と Arduino を繋いで, さらに Arduino にそれを PC に繋いで図 2.5 のようなペアリングの準備をする.

Arduino と USB シリアル変換で通信できているため, その仕組みを応用することでリモートでもログを取れるようにする. ロボット側の Arduino の送信ピンと XBee の受信ピンをつなぎ, データを送信する. 相手の XBee は, 受信側の Arduino にそのまま入れた場合送信ピン同士が競合してしまうため, UART ピンを入力とするソースコードを書くことで強制的に Arduino の機能を停止する. 図 2.6 はプレ大会のログの例である. この図の場合, 左側がオフセットが残っていることで, それがきっかけで振動し始めている. また, センサーにノイズが乗っていて, 赤の出力にまで影響が及んでいることがわかる. 特に微分の値 D はノイズにセンシティブであることがわかる. この場合は D や I を少し増やして, P を少し減らすと安定しそうである.

### XBee の設定

XBee の設定は XCTU というソフトウェアを利用して書き換えた.

まず, ソフトを使ってパラメータを書き換える前にペアリングする両方の MAC アドレスをメモを取っておく. このアドレスはペアリングの際に両方に互いの設定するときを使う. Arduino と XBee を接続する. 接続前に, Arduino はこれらのピンは入力モードにな



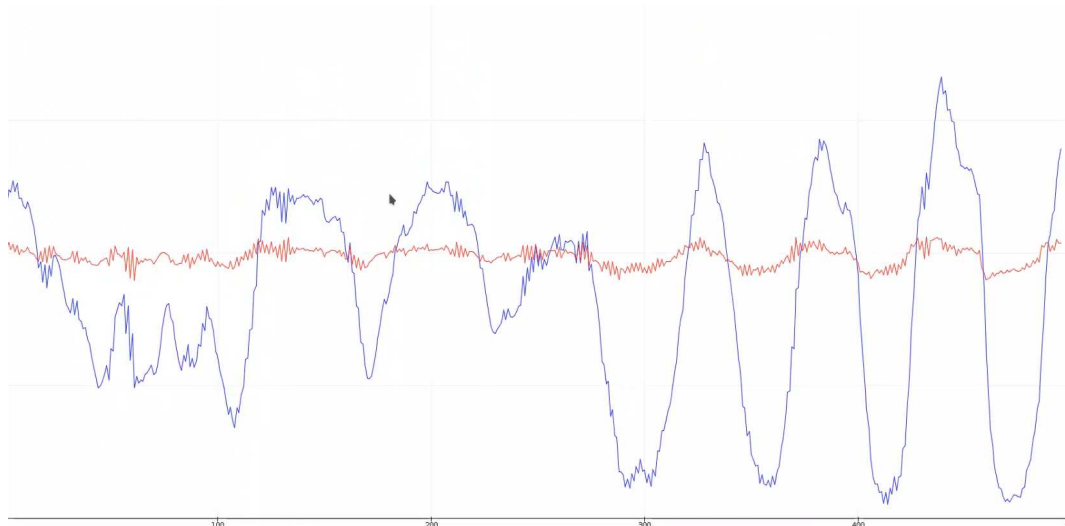


図 2.6: プレ大会でのログ記録例

るようなプログラムが入っていることを確認する．Arduino の Tx ピンを XBee の Rx に，Xbee の Tx を Arduino の Rx につなぐ．

XCTU のソフトウェアを起動し，XBee と接続してファームウェアの更新を行う．接続の際のボーレートは初期出荷時にはボーレートは 9600[bps] になっていた．115200[bps] に書き換えたあとなら必要に応じてボーレートを変更すること．ファームウェアの更新は，ファームウェアの違いにより通信できなくなるのを防ぐために行われる．

ボーレートは 115200[bps] を使用した．Arduino に入っている USB-Serial 変換チップがこのボーレートまでしかサポートしていないためだ．ペアリングするために DH に相手の MAC アドレスの上位ビットを格納し，DL に下位ビットを格納する．最後に書き込みを行えば，ペアリングが可能な状態になる．

ペアリングの確認は，互いの XBee を Arduino とつなぎ，PC から通信できるかどうかを確認すればいい．

## 2.3 ロボットの構成

### 中級用の機体について

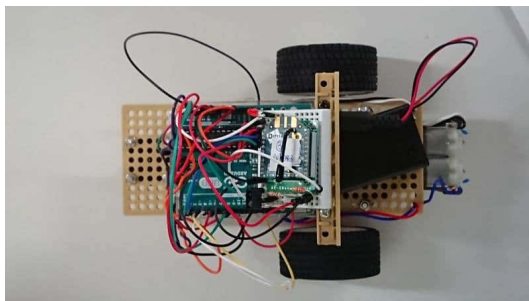
図 2.7 において，図 2.7(a) に中級用の機体を上から見た場合，図 2.7(b) に横から見た場合，図 2.7(c) に下から見た場合を示す．中級用の機体は，走行する速さと走行時の滑らかさに重きを置いて制作した．表 2.1 に示すように，タイヤはスポーツタイヤであり，ギアは標準ギアである．タイヤは機体後部にあり，機体前部はキャスターによって支えられている．

機体下部の 2 つのフォトリフレクタの値を読み取り，その結果から左右のモーターの回転の制御を行っている．機体上部に取り付けられたフォトリフレクタはスタートボタン，発行ダイオードは走行中か否かを示す信号である．発行ダイオードが光っているときは走行中で，光っていないときは未走行である．この機体上部のシステムによって，機体の不備に気づくことが容易になる．例えば，スタートボタンを押しても発行ダイオードが光らず，走行もしない場合はスタートボタンの動作の不備が疑われるが，発行ダイオードが

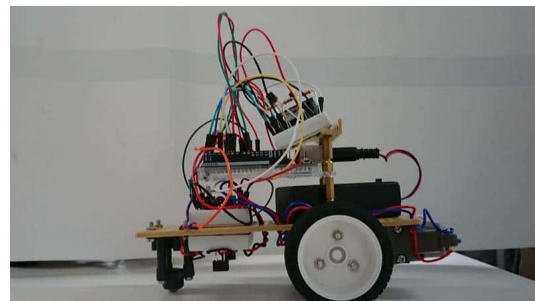
表 2.1: 中級用の機体の構成

タイヤ	スポーツタイヤ
ギア	標準ギア
センサー	2 点
タイヤの位置	機体後部

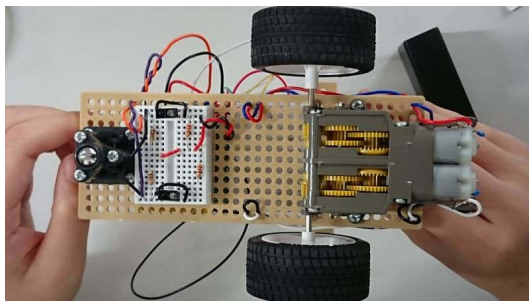
光っているのに走行しない場合はモーター制御部の不備が疑われる。機体上部には XBee モジュールを取り付けてある。これにより機体とコンピュータを有線でつなげていなくても機体走行時のログをコンピュータで取得することが可能になった。中級用の機体は、高速かつ滑らかな走行は得意であるが、高速であるが故に急カーブでは制御が間に合わないことがあった。



(a) 中級用-上から



(b) 中級用-横から



(c) 中級用-下から

図 2.7: 中級用の機体の画像

## 上級用の機体について

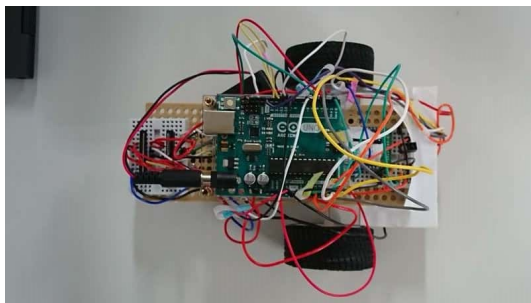
図 2.8 において、図 2.8(a) に上級用の機体を上から見た場合、図 2.8(b) に横から見た場合、図 2.8(c) に下から見た場合を示す。上級用の機体は、コース上を正確に走るというところに重きを置いて制作した。表 2.2 に示すように、中級用の機体との違いは、タイヤが機体前部にあること、フォトリフレクタが 4 つあること、ギアが低速であることである。

ギアを低速にして、タイヤの位置とフォトリフレクタ部の位置を極力近づけることで、急カーブであっても制御を間に合わせることが容易になり、フォトリフレクタ 4 つの値を用いることで安定して走行することが可能になった。上級用の機体は、正確かつ安定な走

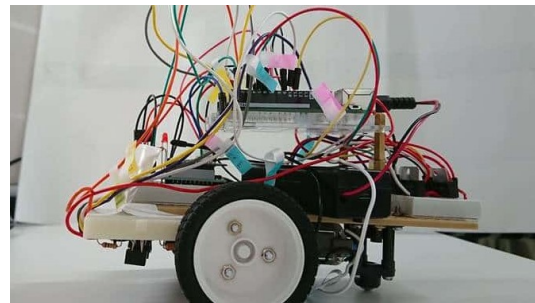
表 2.2: 上級用の機体の構成

タイヤ	スポーツタイヤ
ギア	低速ギア
センサー	4 点
タイヤの位置	機体前部

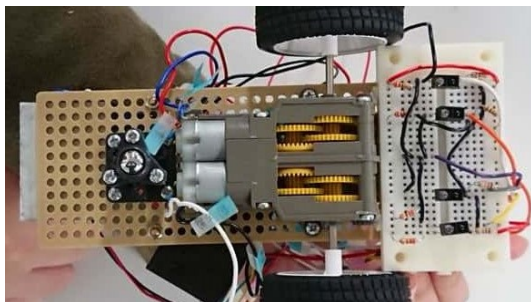
行を得意としているが, 高速にすると制御が間に合わずコースアウトしてしまうことが多かった.



(a) 上級用-上から



(b) 上級用-横から



(c) 上級用-下から

図 2.8: ライントレース上級用機体の画像

## 2.4 ソフトウェア

### 2.4.1 制御方法

まずライントレースをプログラムするにあたって, ハードの制御方法の根本となる制御を pwm 制御, または pid 制御のいずれかを用いるかが重要となる. 以上の二つの制御方法のメリット・デメリットを比較した時, 下図 2.9 のような関係図となる. ここで, ライントレースをするにあたり, カーブが多いコースで安定性かつ, スピードを求めるとなると pid 制御の方が向いていると考えた.

制御概要を図 2.10 に示す. また, pid 制御の詳しい内容については後のセクションで説明する.

## 2つの制御方法比較

	on/off制御	pid制御
・ ソースコードの書きやすさ	○	△
・ 直線の安定度	◎	○
・ カーブにおける 対応力の高さ	△	◎

図 2.9: 二つの制御方法の比較図

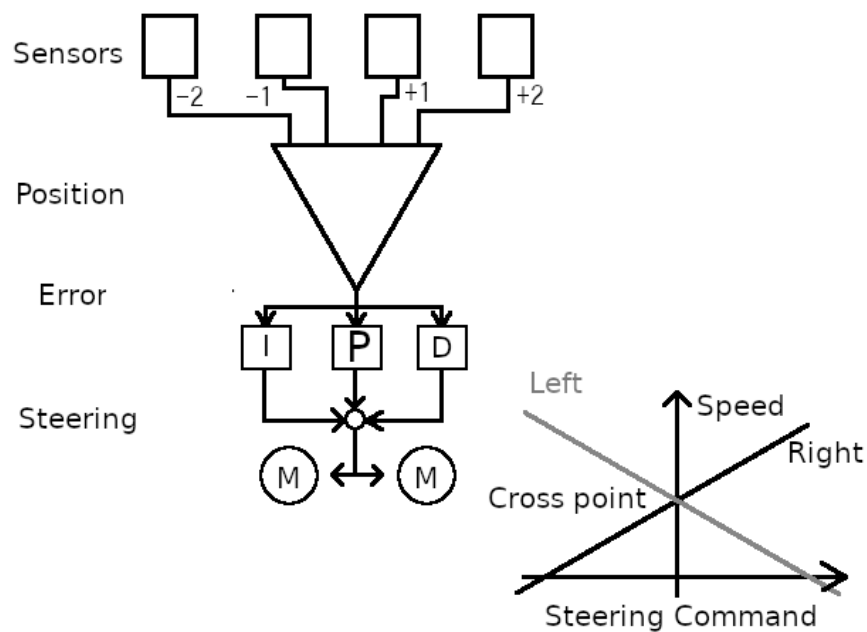


図 2.10: 制御概要

### 2.4.2 基本アルゴリズム

根本となる制御方法を決めた上で、次に考えなくてはならないのはアルゴリズムである。図2.11が、制御の基本フローチャート図である。pid制御を用いるにあたって、ライントレースに用いるセンサを偶数個とし、ハードの真ん中を軸に左右対象にそれぞれ同数個配置し

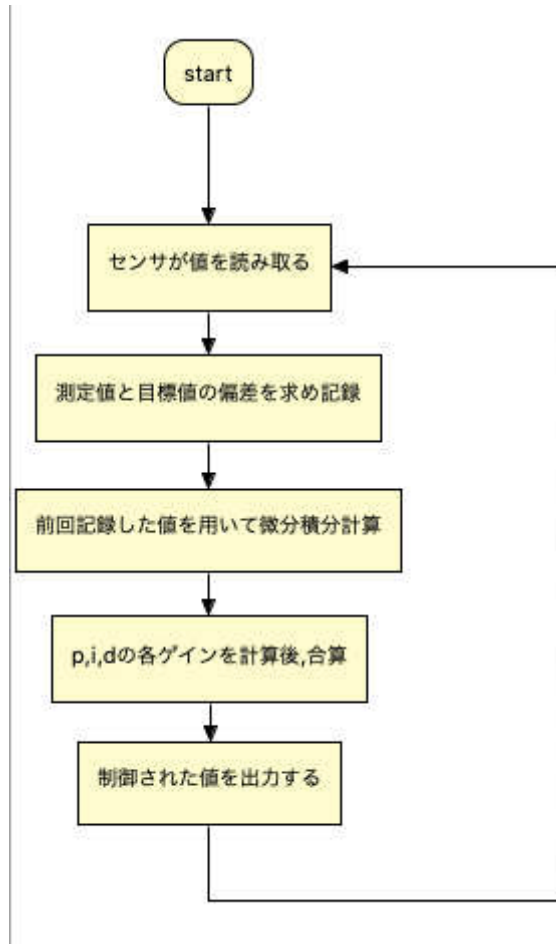


図 2.11: 基本フローチャート図

たものとする。その時,pid 制御で制御したい値を左右のセンサから読み取った値の偏差とし, 目標値を 0 とすることで, ラインの中央を通れるように制御する。ここで, 比例ゲインを  $KP$ , 積分時間を  $KI$ , 微分時間を  $KD$ , 積分区間 (短小時間) を  $\text{delta}-t$  とし, 読み取ったセンサの左右の偏差を  $\text{val}$ , 目標値 (0) を  $\text{target}-\text{val}$  とすると,  $\text{pid}$  制御のソースコードは以下ようになる。

ソースコード 2.1:  $\text{pid}$  制御ソースコード

---

```

1 double pid_control(int val){
2     double p,i,d,pid;
3     static signed long diff[2];
4     static double integral_area;
5     diff[0] = diff[1]; //前回の値の格納
6     diff[1] = val - target-val; //目標値と実際の測定値との偏差
7     integral_area += (diff[0] + diff[1]) / 2.0 * delta-t; //積分
8     p = KP * diff[1];
9     i = KI * integral_area;
10    d = KD * (diff[1] - diff[0]) / DELTA_T;
11    pid = constrain(p+i+d, -255, 255); //-255 <= pid <= 255
12    return pid;
13 }
  
```

---

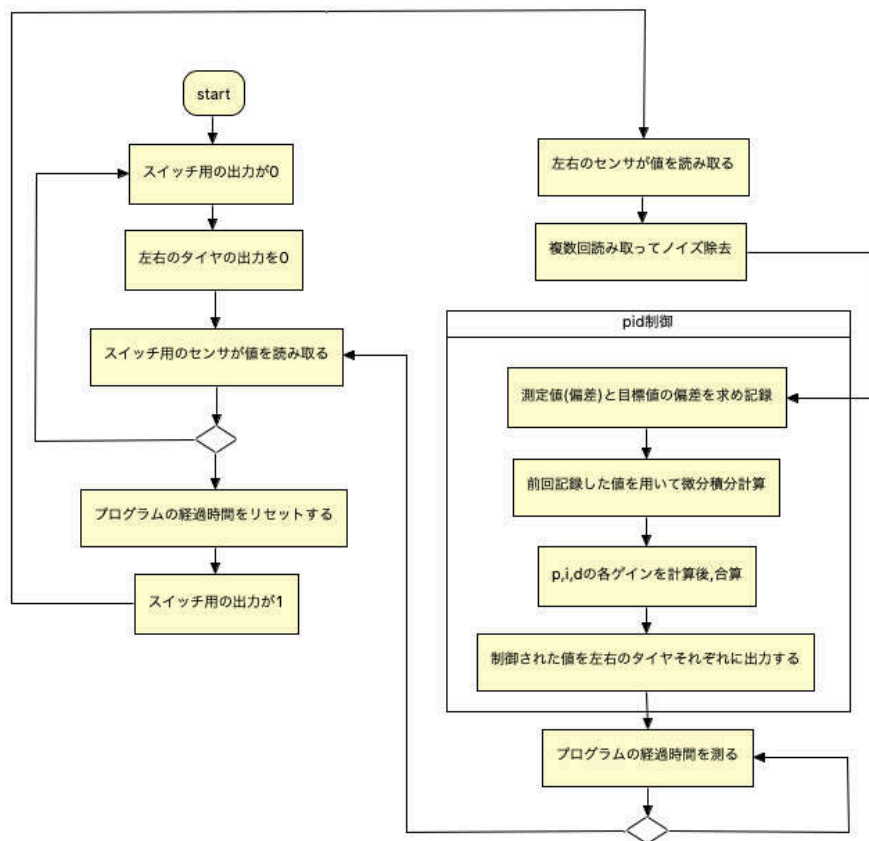


図 2.12: 中級コース用フローチャート図

### 2.4.3 中級用ソースコード

基本アルゴリズムの構成が完了したところで, このアルゴリズムではコースの複雑さに応じて速度の調整ができない. ソースコードの複雑化を避けるためコースに応じてソースコードを分けることにした. 以下は速さを求めた中級用のプログラムとなる. まず, 中級用のコースを分析した際, 直線と緩やかな曲線しかないことがわかる. よって, センサから読み取る情報は最低限の2つあれば十分に曲がり切ることができると判断した.

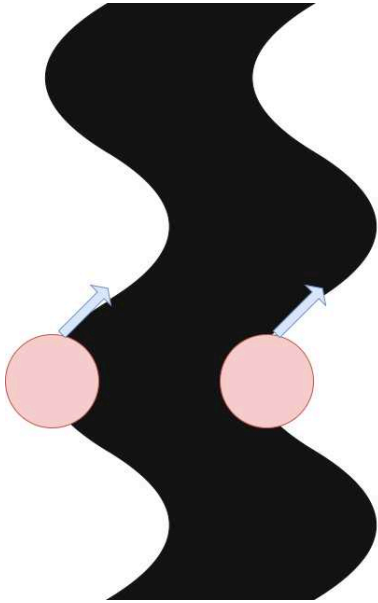
以上のセクションを踏まえた上で, 下図 2.12 が中級用のフローチャートである. このフローチャート図を元に作成したソースコードが 2.6 である.

### 2.4.4 上級用ソースコード

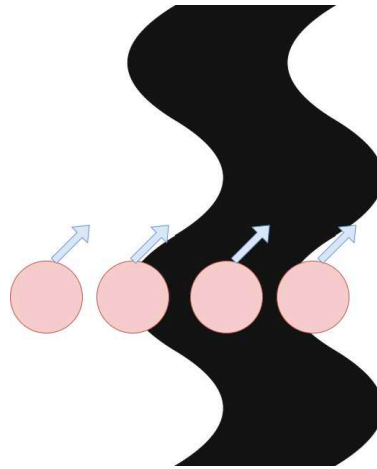
上級コースは基本的に急なカーブが多く, 2つのセンサで読み取れる情報量だけでは不安定な挙動を見せる.

以下, 図を用いて解説する. センサの位置を○, ラインを黒線, 進む方向を青の矢印で表した下図の二つの 2.13(a) と 2.13(b) とを比較したとする. このとき, 厳密にはセンサが読み取る値は円状に広がっているが, センサの位置のみで値を読み取り, pid 制御ではなく, 単純に読み取った値のみで次の動作を決定するものと仮定する. また, センサが読み取る値をセンサ1個分を1とし, 左を負, 右を正として偏差を考える. すると, 2.13(a) では, 偏差の値





(a) センサー 2 個時の読み取り図



(b) センサー 4 個時の読み取り図

がおおよそ 0.3~0.4 と読み取れる。その読み取った値で、図のカーブを曲がるようにプログラミングした場合、図の矢印の向きに曲がろうとする。このとき、先の偏差で曲がるように設定した場合に、次のような急カーブの直後に直線が配置されているコース 2.13(c) を考える。すると、偏差が先ほどと同様に 0.3~0.4 の値で読み取れるので、図のような曲がりかたをする。

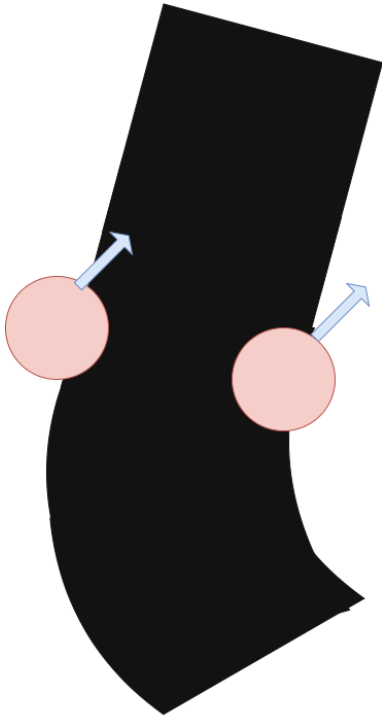
ここで、曲がった後の動作を考えると、2.13(d) のような図になり、偏差はおおよそ -0.8 -0.9 となり、偏差の絶対値が大きくなっている。直線の時の動作において、左右の偏差が徐々に大きくなる、発散するということはいずれ線上から外れることがわかる。よって、ラインレースのプログラムとしては相応しくないことになる。ここで、センサー 4 つの場合を考える。先ほどの 2.13(b) の場合の偏差は 1.0 前後になる。次に、同様の急カーブ直後に直線のコースに、同等の偏差が読み取れるような場合を考えると、下図 2.13(e) のようになる。すると、曲がった後のセンサの配置は下図 2.13(f) になる。この時読み取れる偏差は 0.2 0.3 となり、絶対値が大きくなることはない。また、外側のセンサに 2 倍の重みをつけることで、急カーブライン上に差し掛かった際に、外側のセンサの読み取る値が増えることで、より曲がる際の角度に傾きを持たせることができる。

以上から上級コースの左右のセンサの偏差の定義を変更した。中級コースのソースコードから、センサの読み取る定義を変更し、さらに読み取る時のノイズ除去の試行回数も変更した。ソースコード 2.7 のようになる。

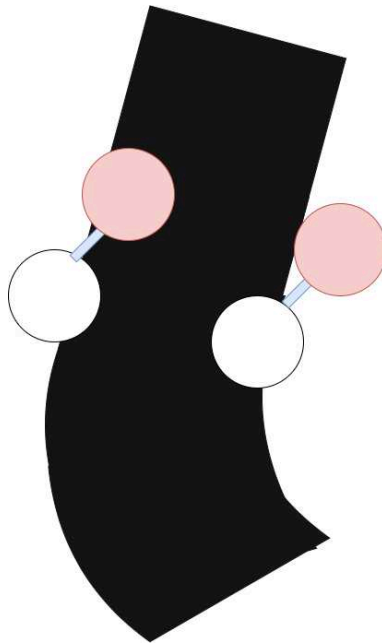
## 2.4.5 PID 制御

### PID 制御について

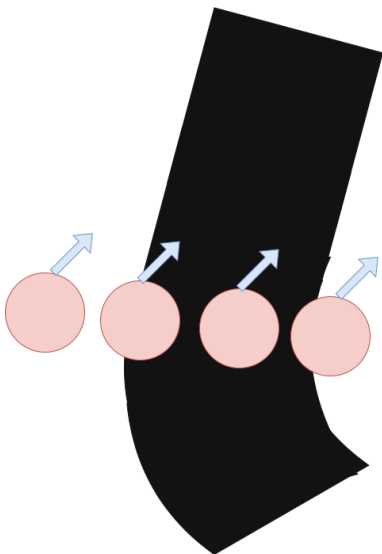
PID コントローラ自体は 3 つのパラメータ PID を除いて、1 入力、1 出力の関数である。コントローラに入力するのは、目標値と制御量との偏差であり、操作量を出力する。PID コントローラの目的としては、安定した制御を実現することが大きな目的であり、例えばエアコ



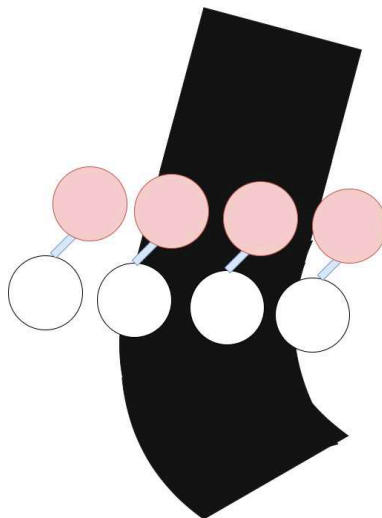
(c) 判定後の進みかた



(d) 判定後の位置



(e) 判定後の進みかた



(f) 判定後の位置

ンの温度調節でも, 部屋を設定温度 (目標値) に素早く近づけて安定させる制御をいち早く実現することができる.

基本的な制御式を示す. 式中  $K_p$  は比例定数,  $K_i$  は積分ゲイン,  $K_d$  は微分ゲイン, 出力を  $u(t)$ , 偏差を  $e(t)$  とする.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

ライントレースのときの目標は, センサー部をライン上中央に配置することである. 良い



PID 制御とは、外乱に対して素早く収束し、振動せず、正確に目標値に到達するような制御である。ライントレーサにおける外乱要素とは、曲がり角である。これはそのまま進むと普通、センサーの値がどんどん偏る、これを外乱と呼ぶ。その外乱に対して素早く応答し、素早く戻さないとコースアウトしてしまう。振動とは、首振りである。目標へ早く到達するというのは、ラインの中央になるべく早く収束するということである。PID コントローラには、P ゲインと I ゲイン、D ゲインがあり、それぞれ別々の役割がある。P ゲインは、偏差にゲイン P をかけたものを操作量にする項であり、基本的な操作量はこの P ゲインにより決定される。

基本的な PID のチューニングはここから始まる。カーブを少しは曲がれるようになるまでこの値を少しずつ増やす。図 2.13(i) は I がいないとき、センサーへの入力、カーブ中で外乱により押し下げられてしまった値を示す。緑の部分が積分（インテグラル成分）で、この値を制御値に加算することでオフセットが相殺される。I ゲインは、P ゲインのみでは、例えばエアコンの出力は、偏差に比例するものしか出ないため、偏差が小さい範囲例えば 1 度のみの違いであれば、それに比例する程度の出力しか出ないため、外乱（冷房をかけているが、外が熱くて中の空気が冷えないなど）に弱い。そこで、I ゲインを追加することで外的要因によるオフセットを減らすことができる。I ゲインは I (インテグラル、積分) 成分を追加することができ、操作量を増やすことができ、それによりオフセットを相殺することができる。

ライントレーサーの場合は、カーブのところ、大きな外乱要素があるが、カーブから少し外れた状態からなるべく早くラインに戻すために使われる。I 成分を増やすと不安定になりやすいのでカーブのときに曲がりきれないとき、オフセットが消えないときに少しずつ増やす。カーブで外乱というのは、カーブでもまっすぐ進もうとしているトレーサーに対して P 制御で曲がろうとしているが、曲がりが連続しているときはその外乱要素が一瞬では終わらずにオフセットとして残ってしまうことを指す。

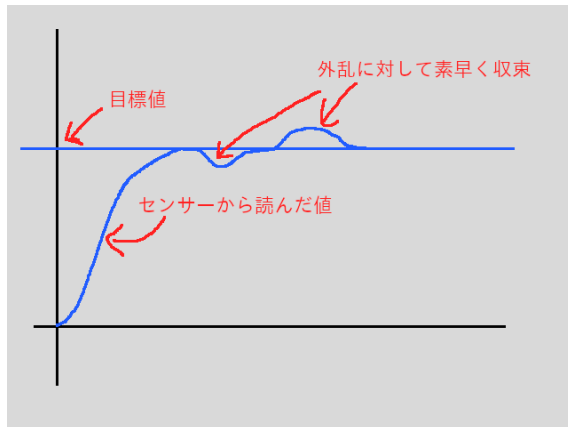
D ゲインは微分成分で、P や I によるオーバーシュートを収束させる働きがある。未来の制御量を予測し、負の操作を行うことで素早く収束させる働きを持つ。D 成分は変化を抑制する方向に制御される。

#### ソースコード 2.2: millis() 関数を用いた実装

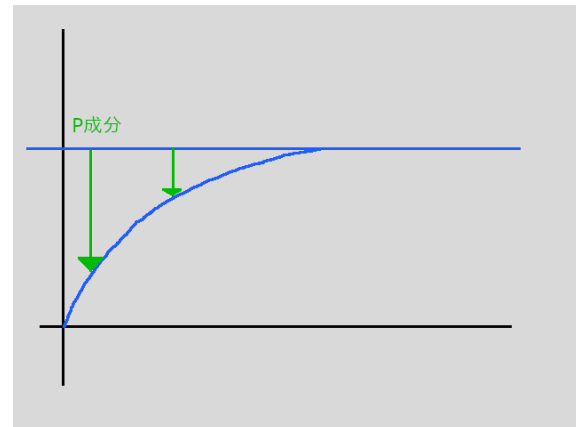
```
1 while(1){
2   //PID 制御
3   while(millis()%10==0);
4   while(millis()%10!=0);//こうすれば、前の処理が間に合えば常に 10ms 間隔になる。
5 }
```

### 2.4.6 on-off スイッチ

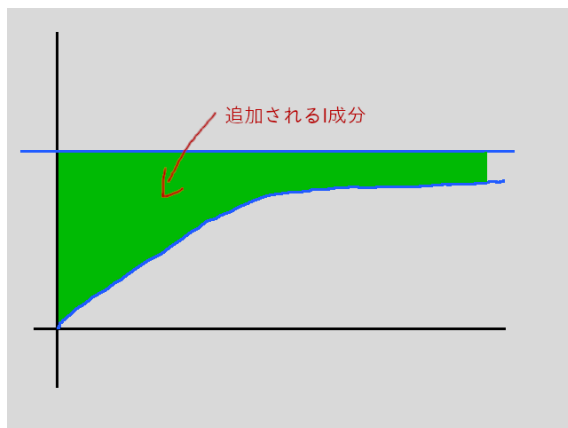
またセンサがラインを読み取る前に、1 つ工夫として、赤外線センサを用いることにより、on-off スイッチ機能を取り付けた。これにより、ハードにソースコードを読み込んだ時の暴走のリスクをカットした。アルゴリズムとして、スイッチ用の赤外線センサ（以下、スイッチセンサとする）が読み取った値が、設定したしきい値 1 を越えたのち、二つ目のしきい値 2 を下回った場合に、センサがタッチされたと判断する。また、そのセンサのタッチ判定によって on-off を切り替える。以下、スイッチセンサを sensor-st とし読み取った値を val-swicth としたソースコードである。



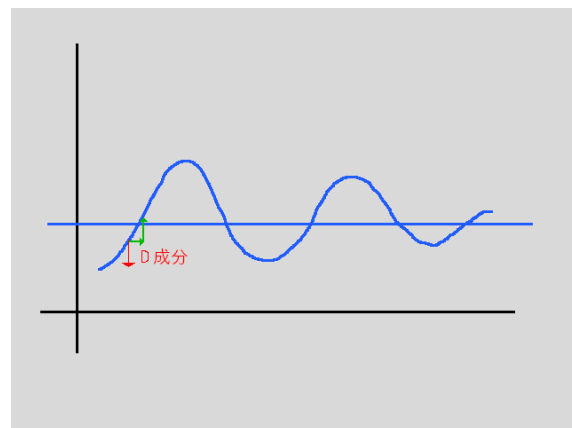
(g) PID1



(h) PID2



(i) PID3



(j) PID4

### ソースコード 2.3: スイッチセンサの実装

```

1 void loop(){ //制御プログラム
2   //センサから値を読み込む
3   while(1){
4     val_switch = analogRead(sensor_st);
5     if (val_switch < 300){
6       while(analogRead(sensor_st)<600);
7       break;
8     }
  }

```

### Arduino における PID 制御の周期

安定した制御周期が PID 制御には欠かせない。delay 関数では処理内容により制御周期が安定しないため、ソースコード 2.2 に示したように、millis() 関数を用いた実装を行う。

### 移動平均を用いたノイズ除去

ソースコード 2.4 に示したように、32 回の移動平均を取ることで、ノイズの除去に成功した。

また、サンプリングを増やすために、ADC のクロックの分周比を下げて analogRead を高速化した。分周比の変更でサンプリング時間を 10 倍程度高速化できる。一方で高速化す

ることに変換の精度は落ちるが、失われる精度は大きくても ADC の 10[bits] 中の 2[LSB] 程度であり、精度の面では電源のノイズのほうが大きいのであまり問題にはならない。

#### ソースコード 2.4: 移動平均を用いた実装

---

```
1 unsigned int scansensor(void){  
2     ADCSRA&=0xFC;  
3     ADCSRA |= 0x04;  
4  
5     int r=0,l=0,r2=0,l2=0;  
6  
7     for(int i=0;i<32;i++){  
8         r+=analogRead(sensor_r)/2;  
9         l+=analogRead(sensor_l)/2;  
10        r2+=analogRead(sensor_r2)/2;  
11        l2+=analogRead(sensor_l2)/2;  
12    }  
13 }  
14 return ((r-l)+2*(r2-l2))/16;  
15 }
```

---

## 2.5 日々の記録

このセクションでは、実験の日々に実際に行ったことの記録を書いておく。この記録は、我々が書いたチーム報告をもとに作成された。

### 2.5.1 10月3日

#### チーム全体の目標

1. チーム全員がライントレースを実現するための方法と、自分の担当タスクを理解すること。
2. モーターとギアボックス、センサーを接続して Arduino からモータードライバを制御できるようにすること。

#### 分担して行ったタスクと目標

最初にメンバーの役割を以下の様に決めた。

PM: 藤井

ハード: 大石, 江原

ソフト: 藤井, 田中, 澤田

次にタスクの割り当てを以下の様にした。

1. ライントレースの方法の調査: 藤井, 田中, 澤田, 江原, 大石
2. ギアボックスの組み立て: 藤井, 大石
3. フォトトランジスタの回路作成: 江原
4. Arduino サンプルプログラムの動作確認: 藤井, 田中, 澤田
5. モータードライバの回路作成: 田中
6. モーターとモータードライバ, Arduino の接続: 田中, 澤田, 江原
7. センサの値を読み取ってモータードライバを制御するプログラム作成: 藤井, 田中, 澤田
8. ハード全体の設計: 大石
9. モーター, ギアボックス, ホイールを組み合わせる: 大石
10. フォトトランジスタの値をプログラムから読み取る: 田中, 澤田

#### 各タスクの達成度

センサー部とモータードライバの接続すること以外の、各タスクの達成度は 100 % である。

## 次回以降の課題

機体は2台作成するが、1台を先に作成しもう1台は先に出来た1台をもとに作成することにした。

先に作成する1台は、センサー部とモータードライバの接続を行い、自走できる状態にする。ライントレースのプログラムの具体的な実装は次週以降の課題である。

## 2.5.2 10月10日

### チーム全体の目標

1. ソフト班はライントレースのプログラムの実装を行う。
2. ハード班は機体を2台とも作成し、それぞれ自走できるようにすること。
3. 1, 2を達成し、ライントレース中級のコースを機体が完走すること。

### 分担して行ったタスクと目標

引き続きメンバーの役割を以下の様に決めた。

PM: 藤井

ハード: 大石, 江原

ソフト: 藤井, 田中, 澤田

次にタスクの割り当てを以下の様にした。

1. フォトトランジスタと Arduino の接続: 大石, 江原
2. モーターと Arduino の接続: 大石, 江原
3. ログ送信用の Xbee はんだづけ: 大石
4. PID 制御で求めた操作量をモーターに伝達する関数作成: 藤井
5. PID 制御の計算関数 (大枠) 作成: 田中
6. 制御後のパラメータ取得環境の整備: 田中
7. Xbee のドライバー等環境設定: 藤井
8. Xbee で2台の Arduino をペアリング: 藤井
9. シリアルモニターで Xbee の接続テスト: 大石
10. PID 制御のプログラムとモーター制御のプログラムをマージ: 藤井, 田中
11. センサーを機体本体に接続: 江原
12. 制御方法の話し合い: 大石, 藤井, 田中
13. p 制御の実装: 田中, 大石

### 各タスクの達成度

Xbee で2台の Arduino をペアリングと、シリアルモニターで Xbee の接続テスト, p 制御の実装, ハード2台目完成以外のタスクは全て達成した。

## 次回以降の課題

機体の制御方法について複数の案が上がったため、選定が必要になった。  
機体の2台目の作成は次週以降も行う。

### 2.5.3 10月17日

#### チーム全体の目標

1. ライントレース初級コースの完走.
2. 機体の制御方法をチーム全体で話し合い, 決定する.
3. PID 制御においてパラメータ調節を行い, より早く滑らかにラインレース出来るようにする.

#### 分担して行ったタスクと目標

引き続きメンバーの役割を以下の様に決めた.

PM: 藤井

ハード: 大石, 江原

ソフト: 藤井, 田中, 澤田

次にタスクの割り当てを以下の様にした.

1. PID 制御で求めた操作量をモーターに伝達する関数作成: 藤井
2. PID 制御の計算関数 (大枠) 作成: 田中
3. 制御後のパラメータ取得環境の整備: 田中
4. Xbee のドライバー等環境設定: 藤井
5. Xbee で2台の Arduino をペアリング: 藤井
6. シリアルモニターで Xbee の接続テスト: 大石
7. PID 制御のプログラムとモーター制御のプログラムをマージ: 藤井, 田中
8. 制御方法の話し合い: 大石, 藤井, 田中
9. p 制御, PID 制御の実装: 田中, 大石, 澤田
10. 機体の試走及び比例ゲイン, 積分ゲイン, 微分ゲインの値の調節: 田中, 澤田
11. Xbee のボーレート変更: 大石
12. 走行の開始・終了を切り替えるスイッチの取り付け: 大石
13. 走行中に店頭する LED の取り付け: 大石
14. ハードウェアの作成: 江原

#### 各タスクの達成度

Xbee で2台の Arduino をペアリングと, シリアルモニターで Xbee の接続テスト, ハード2台目完成以外のタスクは全て達成した.

## 次回以降の課題

Xbee の接続を確立することで PC と機体を無線通信できるようにして、その状態でソースコードの値の変更を機体に渡せるようにする。

中級コースの走行に於いて、何周してもラインからずれないようにする。今回は中級コースを 3, 4 周したところでコースアウトしてしまった。

第 1 回プレゼンテーション大会に向けての資料作成と機体 2 台目の作成については次週以降行う。

## 2.5.4 10 月 24 日

### 今回の目標

- 1, 機体 A について周期の安定化を行い, 中級を安定して 8 秒で走る.
- 2, Xbee を用いて 2 台の Arduino でペアリングを行い, 走行時のデータを取得する.
- 3, 2 台目の Arduino について配線を行い, 自走可能な状態にする.

### チームの目標

今回は以下のようにタスクを分担した.

PM: 藤井

ハード班: 江原, 大石

ソフト班: 藤井, 田中, 澤田

表 2.3 にタスク一覧を示す.

表 2.3: 10 月 24 日のチームタスク一覧

タスク	作業責任者	作業者
機体 A の周期の安定化	田中	大石, 田中, 澤田
機体 A の PID の各値の設定	大石	大石, 田中, 澤田
機体 A の中級の試走	田中	大石, 田中, 澤田
機体 A の上級の試走	澤田	大石, 田中, 澤田
シリアルモニターで Xbee の接続テスト	大石	大石
機体 A と B のペアリング	大石	大石
PID 制御値変更のソフトウェア開発	藤井	藤井
機体 B のソフトウェア開発	藤井	藤井
機体 B のハードウェア開発	江原	江原

## タスクごとの達成度

中級の試走について目標は8秒であったが安定して走ることが出来たのは、10秒であった。上級はきついカーブは曲がりきれない。チーム全体の目標1に対しては80%の達成度である。

目標2に関して、2台のarduinoを用いて通信できたが、arduino間の通信を用いて、PIDの各値を変更するプログラムをデバックするところまでは辿りつかなかったが、Xbeeを用いたペアリングにより走行時のデータを取得することが可能になったので達成度は90%であると言える。目標3に関して、2台目のarduinoについて自走可能な状態にすることができたので達成度は100%であると言える。

## 次回以降の課題

次週はプレ大会である。プレ大会に向けて、2台のarduinoで通信を行うことでPID各値の調整を繰り返す。目標は中級を8秒で走行することである。また、同時進行でプレゼンテーション大会に向けて資料の作成を行う

### 2.5.5 11月7日

#### 今回の課題

- 1, プレゼンの資料を完成させる
- 2, プレ大会の中級コースを1位で走る

#### チームのタスク

今回は以下のようにタスクを分担した。

PM・プレゼン:藤井

ハード班:江原, 大石

ソフト班:田中, 澤田

表 2.4 にチームのタスク一覧を示す。

表 2.4: 11月7日のチームタスク一覧

タスク	作業責任者	作業者
プレゼン資料作成	藤井	藤井, 大石
ハード班:機体 B の配線	江原	江原
ハード班:機体 A のメンテナンス	大石	大石
ソフト班:PID 制御の値調整	田中	田中, 澤田, 大石



## タスクごとの達成度

プレゼン資料は、Google スライドを利用して藤井が完成させた。それに大石を中心として他のメンバーが修正を加えて、発表できる形にした。資料自体はできたので90%の達成度である。

機体 A について、中級コースを 8.48 4 秒で完走する事ができた。ログをシリアルモニターに表示して入出力の発振の様子を監視していたため、5 分という持ち時間の中で効率よく PID 制御のパラメータを変更することができた。まだまだモーターにかける電圧の値を増加させる余地が十分にあるので、値の調整をしながら記録を更新したいと考えている。1 位をとることはできなかったなので、達成度は 90%である。

機体 B には4つのフォトリフレクタを搭載して制御の精度を上げようと試みている。4 センサーを使うに当たって、田中がソフトウェア側の準備を完了させているが、まだ配線が完了していないので、次回には完成させる。達成度は 70%である。

## 次回以降の課題

来週はプレゼン大会 1 があるので、できるだけ高評価を得られるように発表の準備をする。また、ライントレース大会に向けて機体 B の調整を始める。機体 A については、モーターとセンサーの距離を近づける作業から入り、PID 制御の値を再度調整する。

## 2.5.6 11 月 14 日

### 今回の目標

1. プレゼンの完遂
2. 通常作業時に上級コースの制御パラメータの調整
3. 実験最終レポートの書き出し・構成の構築

となる。今回は各班のプレゼン発表が主となるため、通常作業にかけられる時間が極端に短くなってしまうのため、特に成果物があるような目標を設定しなかった。強いて言うなら、目標 3 の最終レポートの外枠である。ひとまず、今回一番おもきをおいたのは目標 1 のプレゼン完遂である。

### チームのタスク

表 2.5 にチームタスク一覧を示す。

## タスクごとの達成度

以下、各タスクの達成度である。

表 2.5: 11 月 14 日のチームタスク一覧

タスク	作業責任者	作業者
プレゼン資料編集・ミーティング (講義前)	大石	藤井, 大石, 澤田, 江原
GitLab でのグループ作成	澤田	澤田, 江原
実験最終レポートの作成開始	藤井	藤井, 澤田, 江原
上級コース用のパラメータ調整・コード変更	田中	田中, 大石

#### 1. プレゼン資料編集・ミーティング:

本日の講義前にて全員で集まり、ミーティング兼編集作業を行った。資料の調整や質疑応答の予想とその対策を行った。スライドとしても見やすく、かつ内容が論理的でしっかりとしたプレゼン資料となった。作業達成率として、実際にプレゼンを行った時の手応えとして十二分に感じられた。

現に、プレゼンの質疑応答時にも、ミス指摘されるような質疑はされなかった上に、質疑をされた時に対策通りアピールポイントをしっかりと付け加えることができた。さらに、プレゼン後に他の班員にプレゼンの感想を求めたところ、好評だった。目標 1 の達成度としても客観的に見ても、胸を張って 100 % であると言える。

#### 2. GitLab でのグループ作成:

このタスクは最終レポートを作成するにあたって、全員でレポートを共同作成することを見越し、GitHub でレポート用の Tex を共有した方がいいと考えた。しかし、個人無料ユーザ用の GitHub では共有できる人数の制限が厳しいという問題点があった。その問題点を解決するために、我々は GitLab を利用することにした。

GitHub でのプライベートリポジトリにおける共同編集は、今年の頭に無料版においても解放されたが、3 人までしかグループとして作れないが、グルーピングの機能に長けている GitLab ならば GitHub と UI の違いはあれど、リポジトリ管理の性能として問題なく使える。前置きが長くなってしまったが、GitLab を利用するにあたって、各班員がアカウントを作り、そしてグループを作成する必要がある。以上のことを踏まえ、実際に各班員でアカウントを作成し、それをグルーピングするまで行った。達成度 100 % である。

#### 3. 実験レポートの作成:

タスク 2 の作業の延長となるが、実験レポートを Tex で作成し始めた。作業時間が短かったため、構成を構築し外枠を完成させるには至ったが、まだ試作段階であるため次回以降に随時修正・追加していく必要がある。達成度としては作業のスタートをタスクとしたため、達成には至っているわけだが、作業全体におけるペースとしての達成度は 8 割前後である。目標 3 はタスク 2 と合わせても概ね達成されていると言える。

#### 4. 上級コース用の機体のパラメータ調整・ソースコード変更:

今回の実験において、このタスクの作業進行度が一番低かった。結論からいうと、作業結果として内容に進展があまりなかった。

このタスクの事前に設定した具体的な内容として、ソースコードを上級用(中級コース用とハードを分けている)に調整し、実際に幾度か走らせることで各ゲインや出力などの

パラメータデータをえて、そのデータを参考にパラメータ調整に取り組むに至る(パラメータにおおよそのあたりをつける)ことが目標であった。確かに、ハードを変更することで、そのハードの様々な特性を考慮したソースコードに書き換えなくてはいけないのはわかっていたため、時間が多少取られることも想定していた。しかし、想定と違い、ハードの特性が少し難解であったこと。それに加えて通信機器の不調が重なり、全く走行データのログが取れなかった。

この9班は、他班と違って走行ログを通信機器でデータ化し、可視化することが他班とは違う強みになっているわけだが、その強みを活かせることができない、ということになってしまった。それならばと、走行の様子を観察する形で、原始的に走行データを分析しようと考えたが、ここでさらにボードの不調により書き込みに膨大な時間がかかるように。ひとまず中級用のコードをもとに、センサ4つでそれらしい制御で走行することはできたが、各パラメータゲインの適切な値に全くあたりをつけることができなかった。プレ大会の準備に時間を大きく割いてしまい、時間に余裕がないため、来週から本番の大会まではこの続きの作業が一番の課題となってくる。達成度としては6割弱といったところだろう。

## 次回以降の課題

達成度報告でも述べたように、我々9班には時間に余裕がないので、上級用の走行調整に最も時間と人数をかけなければならない。今回の作業の反省から考えて、ハードの見直し・修正、ソースコードのデータ取得によるPIDゲインのパラメータ調整、より複雑なコースに対応できるif分岐ソースコードの調整の3つが大きなタスクとなるだろう。

また、5人全員がこの3つのタスクを分担するのも効率が悪くなってしまうので、今回同様、レポート作成にもタスクを分割する可能性もある。

ソースコードの変更にあたって、今まではメッセージ機能を搭載したSNS、Slackでソースコードの変更を随時加えていたが、見づらい上に、どれがどのソースコードで何が最新版なのか分かりづらかったため、GitLabでソースコードを共有化することにした。これにより、講義時間外での作業もより共有しやすくなり、時間が足りないという問題点に対しても、時間外活動のさらなる活発化という形で対応することができる。

## 2.5.7 11月21日

### 今回の目標

上級を走る機体の完成と上級の完走。レポートの第1章のプロジェクトの作成とライントレースに関する技術的部分の作成

### チームのタスク

- 1, レポート作成[プロジェクト] 藤井
- 2, レポート作成[その他] 澤田
- 3, 上級のハードウェア作成 大石, 江原
- 4, バックの実装 田中

## タスクごとの達成度

1, レポート作成 [プロジェクト] 60%ほど書き上げることができた. gitlab を使って tex のファイルを共有することでレポートをスムーズに分担して書くことができています.

2, レポート作成 [その他] 現在まとめられている技術に関しては全て書ききることができたがそれ以外の部分はまだであるので, 50 % ほどであると言える.

3, 上級ハードウェア作成 上級コースを 2 周走りきる機体を作成することができたので 100 % である. しかし, タイヤのコンディションによってうまく走ったり走らなかったりしているので, 今後も微調整が必要な部分であると言える.

4, バックの実装技術的な問題点 (センサーの位置など) によってバックを実装するのをやめたので, バックをせずに上級コースを走りきれる調整方法を模索中である.

## 次回以降の課題

1, レポート作成についてはスムーズに分担して書くことができる仕組みになっているので, 今後ともに班別作業の際には手が空いた人はレポートを進めていく予定である.

2, 上級コースを走れてはいるものの, 上記の通り, 未だ不安定な状態であることは間違いないので, 来週の班別作業でも調整作業を進めていく必要がある.

## 2.5.8 11 月 28 日

### 今回の目標

2 台の機体それぞれの調整及び各コースの完走レポート執筆

### チームのタスク

1. レポート作成 藤井 澤田
2. 上級用の機体の調整 大石 田中
3. 中級用の機体の調整 藤井 澤田 江原

## タスクごとの達成度

1 のレポートの達成度は 50% である. 機体の調整の合間に行ったため文章としてまとめられたものは少なかった.

2, および 3 の達成度はそれぞれ 50% と 90% である. 2 台とも完走は出来たが, より大きな速度での完走を目指していたためこの評価になった.

## 次回以降の課題

レポートについては, 班別作業の時間に進める. 機体については, 次の追従大会に向けて改造を行う. 先導機体のどこにアクリル板を付けるかや追従機体の追従アルゴリズムについてチーム全員で方針を話し合い, 作業に取り組む予定である.

## 2.6 大会の考察

### 2.6.1 プレ大会

#### プレ大会について

プレ大会では，初級，中級の2コースが用意された．我々は，中級コースを最速タイムで走ることを目標として掲げた．

#### プレ大会の結果

プレ大会は，中級コースを完走し，8.48 秒の記録を出すことができた．これは，1 位の 8.12 秒と 0.36 秒差で惜しくも 2 位であった．

#### プレ大会の考察

速度を落とした状態で PID 制御を行い，走行可能になったら速度を上げて再調整という手法をとっていたのだが，これでは二度手間になってしまい時間を有効活用して，調整することができなかった．また，1 位になったチームと比べてモーターの出力を強くした場合の PID の制御値がうまく調整出来なかったことが原因といえる．

### 2.6.2 本大会

#### 本大会について

本大会では，初級，中級，上級の2コースが用意された．我々は，上級コースを最速タイムで走ることを目標として掲げた．

#### 本大会の結果

本大会では，上級コースを 82.820 秒で走り，16 班中 10 位という結果であった．

#### 本大会の考察

上級コースの完走を目指し，上級機体を作成したが速度を上げて PID 制御を行うことがうまくいかずゆっくり走ることしかできなかった．

### 2.6.3 追従大会に向けて

追従大会では，プレ大会の反省を生かしたい．追従大会では，先導機体と追従機体の2体のバランスの取れた調整が必要不可欠である．先ほどの考察で上げた通り，遅いスピードで調整していくと速度を上げた際に二度手間になってしまう．最初から理想のスピードで調整することで時間を有効活用して調整していきたい．

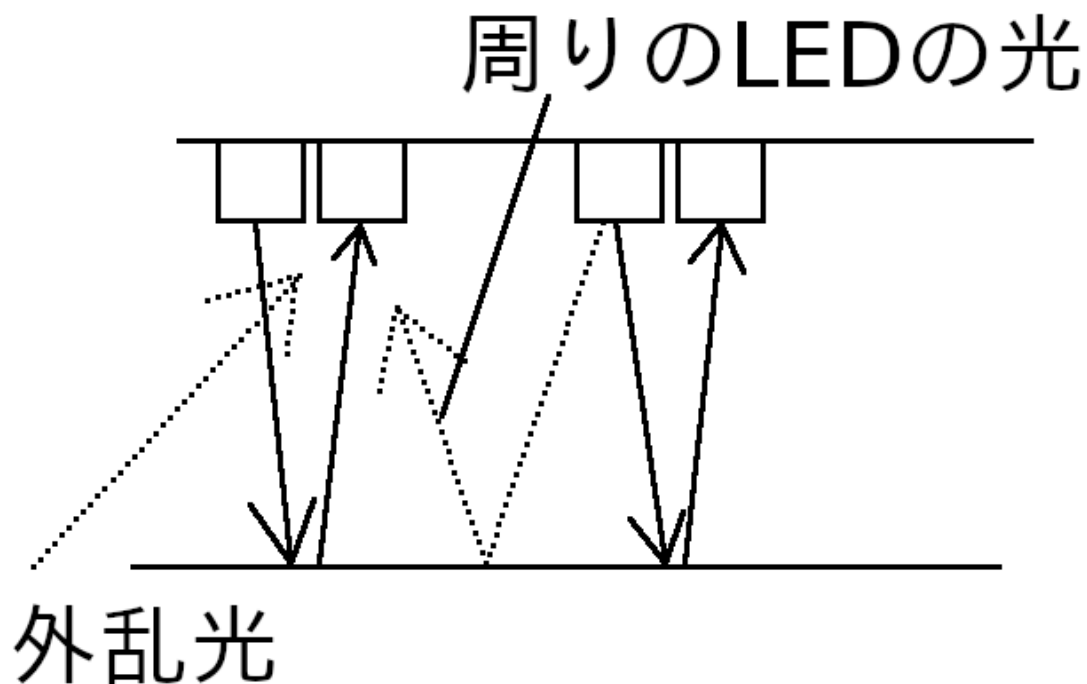


図 2.13: ラインスキャンに関する外乱

## 2.7 今後の展望

### 2.7.1 センサの改良

今回の敗因の一つに、センサの取得が不安定になったことによる再現性のない走行不良が挙げられる。先週動いていたが今週動かないということが多かったからである。

この理由の一つに、センサのオフセットの変動と、周りのセンサの影響を受けやすいセンサ回路が挙げられる。センサ回路は、フォトリフレクタのアレイからなり、このアレイがセンシングするときに受けるオフセットの影響を考慮していなかった。

センサアレイが受ける外乱の要因として、外乱光や周辺の LED による測定点のズレが挙げられる (図 2.13)。

#### 同期検波によるセンシング

フォトリフレクタの LED を動的に制御し、これらの外乱の影響を最小限に抑える仕組みがある (図 2.14)[2]。

まず、対象とするセンサ周辺の LED の光が回り込んでしまう問題は、対象とする LED 以外をオフにすればいい。外乱光はなるべく入らないように作ることも可能だが、外乱光が一定だと考えれば、LED を消灯したときと点灯したときのセンサの値を引く。これによりセンサのばらつきをさらに抑制することができる。

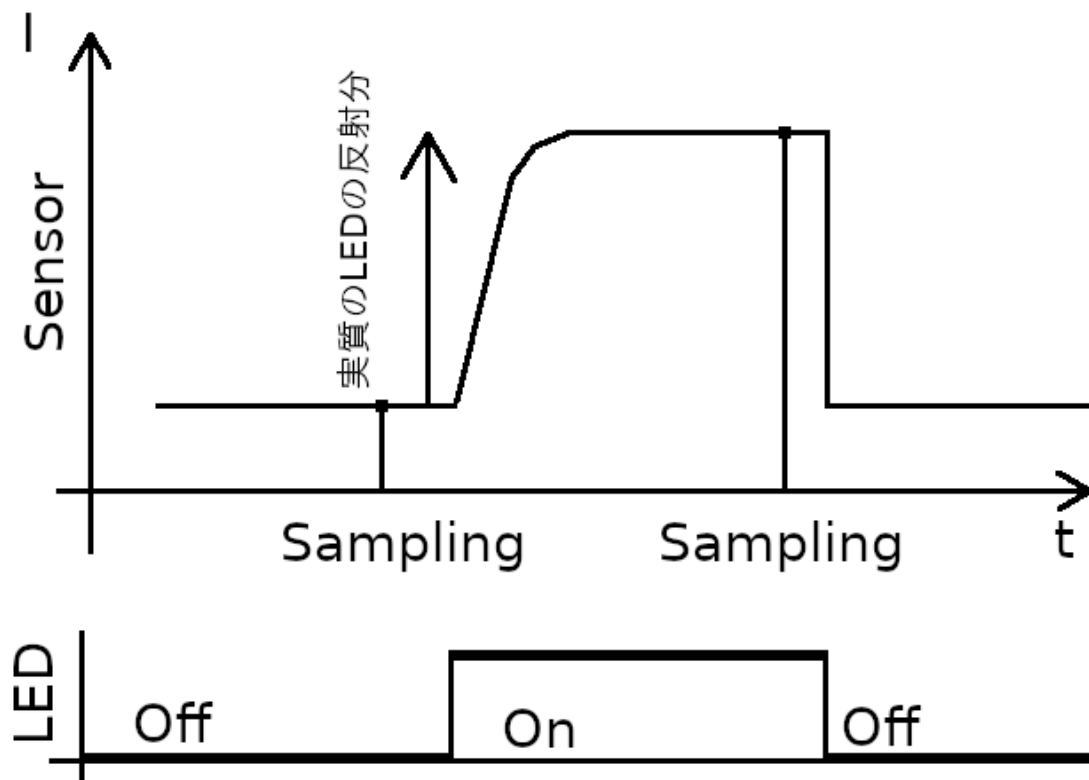


図 2.14: 同期検波方式

## ブラックとホワイト，位置の補正

センサの出力は，それぞれ別々のバイアスやオフセットを持っている．さらにセンサの出力は，照射範囲が円形であり，この面積を黒い成分の分だけ積分すると一般にセンサから得られる情報は非線形特性を持っている．このとき，リニアリティに補間して制御することは，線形システムの性質からPID制御の要件を満たさず，あまり良くない．

そこで，最初にセンサの値と位置の関係を補正したテーブルを持つておくことで，これらの特性を改善することができる．例えば，ラインの上で超信地旋回による首振り動作をして，そこで得られるセンサの値の逆関数を求めてセンサの値を補正することが考えられる．

その他の実装方法として，より簡単にするためにはホワイトとブラックの値を自動的に取得し，円形の照射範囲を仮定して逆三角関数から領域を推定することも考えられる．

## 2.7.2 定数調節の実装

P, I, D のゲインの調節を行う方法にはいくつかある．

## ハードコーディング

今回採用した方式で、プログラム内部に P, I, D ゲインを埋め込む方式である。一番確実であるが、調節のたびに再コンパイルとプログラムの転送が必要になり、非効率である。

## 半固定抵抗による方式

半固定抵抗で P, I, D ゲインを調節する方式である。ある程度の範囲を決め打ちして、細かい値を半固定抵抗から読み取る。読むときはリセット時などある決められたときに一回だけ読むようにすることで、走行中のノイズで予期せぬ書き換えを防ぐことができる。

欠点としては、パラメータ数だけ Analog ポートが必要となる点と、半固定抵抗が予期せず回ってしまうとせっかくの調整済み定数がずれてしまう点である。今回は、ポートの数が足りず、実装できなかった。調整した定数をシリアル上で読み出すことで、うまく行ったときの定数をハードコーディングする方法も考えられる。

## リモートによる書き換え

PID 制御を実現するためのパラメータ KI, KP, KD は頻繁に変更が生じる。ところが、変更を行うたびに Arduino へプログラムをコンパイルして書き出すのは非常に効率が悪い。そこで、XBee を用いてシリアルモニターから KI, KP, KD の値を変更できるプログラムを作成した。

## 概要

2.15 のフローチャートを用いて動作を説明する。まず、コマンド 999 が入力されるまで待機する。999 が入力されたら、値の更新を監視する。なぜなら、シリアルモニターの値読み込みは loop() 関数から常に呼ばれ続け、一度値を入力するとその値が延々と読み込まれるからである。入力が更新されていたら、コマンド 666 と一致するか検査する。666 が入力されたら、KI, KP, KD の順に書き換えるパラメータを変える。もしシリアルの入力が 666 以外だったら、その値を現在変更対象であるパラメータに書き換える。

また、入力する値の仕様は以下の通りである。

- 入力が許されるのは整数値のみである。
- 入力値は、一桁目を小数第 1 位とする double 型の値として処理される。
- 0.999, 0.666 を KI, KP, KD の値として設定することはできない。

## 実行例

機体のスイッチを入れてから、シリアルで KI, KP, KD の値をそれぞれ 0.05, 0.07, 0.08 に変更する例を 2.15 に示す。この実行の結果、Arduino 上の EEPROM に値が書き込まれる。



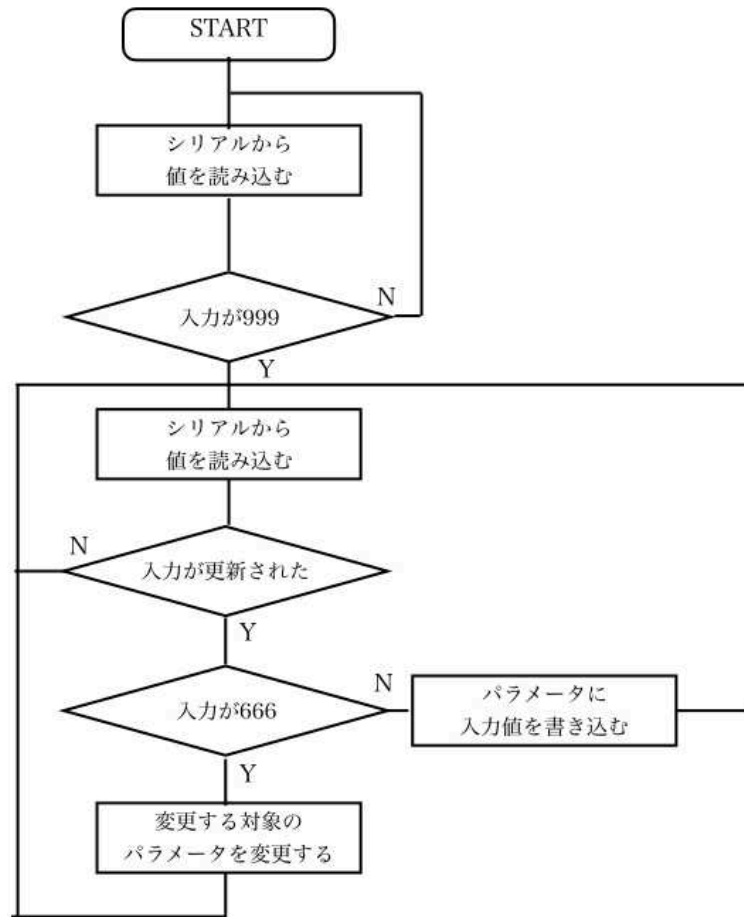


図 2.15: パラメータ変更プログラムのフローチャート

## シリアルモニターからの入出力

まず, シリアルモニターからの値を入力する関数 `ReadSerialInput` について説明する. この関数では, 入力データが溜まるまで `delay` 関数を用いて待機した後, 入力データを 1 桁ずつ `byte` 型配列に格納する. その配列を `ConvertSerialInputToData` 関数に渡して, 小数値に変換している. `ConvertSerialInputToData` 関数は, 第一引数の `byte` 型配列の 0 番目を小数第 1 位, 1 番目を小数第 2 位... とみなした `double` 型の値を返す関数である.

EEPROM に書き込む時はメモリの番地を意識しなければならない. その機能を関数として切り分けた部分が 2.5 である.

### ソースコード 2.5: EEPROM に `double` 型を書き込む関数

```

1 void EEPROM_writeDouble(int ee, double value)
2 {
3     byte* p = (byte*)(void*)&value;
4     for (int i = 0; i < sizeof(value); i++)
5         EEPROM.write(ee++, *p++);
6 }
7
8 double EEPROM_readDouble(int ee)
9 {
10    double value = 0.0;
11    byte* p = (byte*)(void*)&value;
12    for (int i = 0; i < sizeof(value); i++)

```



図 2.16: 実行の例

```

13     *p++ = EEPROM.read(ee++);
14     return value;
15 }

```

結果として、シリアルモニターから PID 制御のパラメータを変更し EEPROM に保存しておいて再現することが出来た。しかし、リモートで PID の値を変更することはできなかった。理由は 2 台の Arduino を接続した状態でシリアルモニターを起動し値を入力することが出来なかったためである。よって PID コントローラの調整でこの機能が役立つことはなかった。

### 2.7.3 ラインセンサの配置の検討

ラインセンサは、横に並べる方式が主流ではあるが、横に並べるだけでは直線検出ができず、速度を出していいのか出さないほうがいいのかがわからない。そのため、縦にもセンサを並べることで、曲がり角が近くにあるかまだ直線の範囲なのかが判定でき、より高速にラインレースできると考えられる。

### 2.7.4 バッテリー電圧の読み出し

バッテリーの残量によって走行が不安定になることがある。そのため、バッテリー電圧  $V_{in}$  を分圧し、Arduino で読み出してその内容をフィードバックすることで安定した走行

につなげることができそうである。

### 2.7.5 センサの電源の安定化

センサ電源は、5[V] から取っているが、できれば 3.3[V] などから取りたかった。これは、センサに乗るノイズがかなり大きくてセンサの読み出しに悪影響を与えていることがわかったからである。5[V] ラインはモータへ出力するとかなり変動するが、3.3[V] は 5[V] 系統の電圧をさらにレギュレータにより降圧するため比較的安定している。これらを使うのにはメリットが大きい。

一方で低電圧化することで 5[V] で調節していたパラメータを変更する必要があり、そのコストを考えて今回は実装していない。

### 2.7.6 基板の使用

基板を用いた配線を進めてブレッドボードを減らすことで配線が抜けることや高密度なセンシングを実現したかった。また、直線検出は、ブレッドボードによるレイアウトの制約を強く受けるため、大きめの基板で実装することで直線検出しやすい状況になると考える。ブレッドボードは重量があるため、軽量化にもつながる。

## 2.8 ライントレーサのまとめ

ライントレーサ大会では、上級コースを完走することができた。基本的なトレースに関する知識、プロジェクトマネジメントに関する知識などを得ることができた。しかし、調整時間が足りず、ラップタイムは最下位となってしまった。

追従大会に向けては、ラップタイムを縮めるようにライントレース能力を向上させるとともに、プロトタイピングモデルのもと、追従機体の迅速なプロトタイプ制作を目標とする。

## ソースコード

### ソースコード 2.6: 中級ソースコード

---

```
1 //Arduino Uno のピン配置
2 //右モーター
3 const int motor_r1 = 3;//Arduino の 3 番ピン
4 const int motor_r2 = 5;
5 //左モーター
6 const int motor_l1 = 6;
7 const int motor_l2 = 9;
8 //フォトリフレクタ
9 const int sensor_r = A3;
10 const int sensor_l = A1;
11 //const int sensor_r2 = A4;
12 //const int sensor_l2 = A5;
13 const int sensor_st = A2;
14 //LED
15 const int LED = A0;
16 //センサから読み取った値 (analog)
17 int val1 = 0;
18 int val2 = 0;
19 //int val3 = 0;
20 //int val4 = 0;
21 int val_switch = 0;
22
23 //左右のセンサの差分
24 int val_c = 0;
25 //PID 制御された値
26 double power_r = 0.0;
27 double power_l = 0.0;
28 double power_pid = 0.0;
29 //目標値
30 #define target_val 0.0
31 //比例ゲイン
32 #define KP 0.0670
33 //積分時間
34 #define KI 0.008
35 //微分時間
36 #define KD 0.0035
37 //短小時間
38 #define DELTA_T 0.010
39 //直線速度
40 #define AVERAGE 85
41 #define balance_r 0.85
42 #define balance_l 1.15
43
44 //PID 制御する関数
45 double pid_control(int val){
46     double p,i,d,pid;
47     static signed long diff[2];
48     static double integral_area;
49     diff[0] = diff[1]; //前回の偏差の値の格納
50     diff[1] = val - target_val; //目標値と実際の測定値の偏差
51     integral_area += (diff[0] + diff[1]) / 2.0 * DELTA_T; //積分
52     p = KP * diff[1];
53     i = KI * integral_area;
54     d = KD * (diff[1] - diff[0])/DELTA_T;
55     pid = constrain(p+i+d, -255, 255); //-255 <= pid <= 255
56     return pid;
```

```

57 }
58
59 //右モータの動作
60 void run_Right(int n , int power){
61     switch(n){
62         case 0://後進
63             digitalWrite(motor_r1, LOW);
64             analogWrite(motor_r2, power);
65             break;
66         case 1://前進
67             analogWrite(motor_r1, power);
68             digitalWrite(motor_r2, LOW);
69             break;
70         case 2://ブレーキ
71             analogWrite(motor_r1, 0);
72             digitalWrite(motor_r2, LOW);
73             break;
74     }
75 }
76 //左モータの動作
77 void run_Left(int n , int power){
78     switch(n){
79         case 0://後進
80             digitalWrite(motor_l1, LOW);
81             analogWrite(motor_l2, power);
82             break;
83         case 1://前進
84             analogWrite(motor_l1, power);
85             digitalWrite(motor_l2, LOW);
86             break;
87         case 2://ブレーキ
88             analogWrite(motor_l1, 0);
89             digitalWrite(motor_l2, LOW);
90             break;
91     }
92 }
93
94 void setup(){ //実行時に 1回のみ実行
95     //それぞれの motor に対するピンを出力ポートに設定
96     pinMode(motor_r1, OUTPUT);
97     pinMode(motor_r2, OUTPUT);
98     pinMode(motor_l1, OUTPUT);
99     pinMode(motor_l2, OUTPUT);
100     Serial.begin(115200);
101     Serial.println("start");
102 }
103
104 void loop(){ //制御プログラム
105     //センサから値を読み込む
106     while(1){
107         val_switch = analogRead(sensor_st);
108         //Serial.print("waiting\n");
109         //Serial.print(val_switch);
110         if (val_switch < 130){
111             while(analogRead(sensor_st)<300);
112             break;
113         }
114
115         run_Right(1,0);
116         run_Left(1,0);

```

```

117     digitalWrite(LED,LOW);
118 }
119 while(1){
120     digitalWrite(LED,HIGH);
121     val1 = (analogRead(sensor_r)+analogRead(sensor_r)+analogRead(sensor_r)+
122             analogRead(sensor_r))/4;
123     val2 = (analogRead(sensor_l)+analogRead(sensor_l)+analogRead(sensor_l)+
124             analogRead(sensor_l))/4;
125     //val3 = analogRead(sensor_r2);
126     //val4 = analogRead(sensor_l2);
127     val_c = val1 - val2; //+ 2*(val3-val4);
128     //読み取った値を PID 制御する
129     power_pid = pid_control(val_c);
130     //シリアルボードに制御した値を表示させる
131     Serial.print(val1);
132     Serial.print(",");
133     Serial.println(val2);
134     //モータを左右でそれぞれ実行する
135     //power_r = balance_r*(AVERAGE + power_pid);
136     //power_l = balance_l*(AVERAGE - power_pid);
137     //run_Right(1,power_r);
138     //run_Left(1,power_l);
139     //delay(10);
140     while(millis()%10==0);
141     while(millis()%10!=0);
142     val_switch = analogRead(sensor_st);
143     if(val_switch < 130){
144         while(analogRead(sensor_st)<300);
145         break;
146     }
147 }
148 }
149 }

```

---

## ソースコード 2.7: 上級ソースコード

---

```

1 //Arduino Uno のピン配置
2 //右モーター
3 const int motor_r1 = 6;//Arduino の 3 番ピン
4 const int motor_r2 = 9;
5 //左モーター
6 const int motor_l1 = 3;
7 const int motor_l2 = 5;
8 //pwm 用のピン
9 const int pwm_motor_r = 11;
10 const int pwm_motor_l = 10;
11 //フォトリフレクタ
12 const int sensor_r = A3;
13 const int sensor_l = A1;
14 const int sensor_r2 = A4;
15 const int sensor_l2 = A5;
16 const int sensor_st = A2;
17 //LED
18 const int LED = A0;
19 //センサから読み取った値 (analog)
20 int val1 = 0;
21 int val2 = 0;
22 int val3 = 0;
23 int val4 = 0;
24 int val_switch = 0;
25

```

```

26 //左右のセンサの差分
27 int val_c = 0;
28 //PID 制御された値
29 double power_r = 0.0;
30 double power_l = 0.0;
31 double power_pid = 0.0;
32 //目標値
33 #define target_val 0.0
34 //比例ゲイン
35 #define KP 0.035
36 //積分時間
37 #define KI 0.0027
38 //微分時間
39 #define KD 0.0040
40 //短小時間
41 #define DELTA_T 0.010
42 //直線速度
43 #define AVERAGE 65
44 #define balance_r 1.0
45 #define balance_l 1.0
46 //脱線判定
47 #define border 80
48
49 //PID 制御する関数
50 double pid_control(int val){
51     double p,i,d,pid;
52     static signed long diff[2];
53     static double integral_area;
54     diff[0] = diff[1]; //前回の偏差の値の格納
55     diff[1] = val - target_val; //目標値と実際の測定値の偏差
56     integral_area += (diff[0] + diff[1]) / 2.0 * DELTA_T; //積分
57     p = KP * diff[1];
58     i = KI * integral_area;
59     d = KD * (diff[1] - diff[0])/DELTA_T;
60     pid = constrain(p+i+d, -AVERAGE, AVERAGE); //-AVERAGE <= pid <=
        AVERAGE
61     return pid;
62 }
63
64 //右モータの動作
65 void run_Right(int n , int power){
66     switch(n){
67         case 0://後進
68             digitalWrite(motor_r1, LOW);
69             digitalWrite(motor_r2, HIGH);
70             analogWrite(pwm_motor_r,power);
71             break;
72         case 1://前進
73             digitalWrite(motor_r1, HIGH);
74             digitalWrite(motor_r2, LOW);
75             analogWrite(pwm_motor_r,power);
76             break;
77         case 2://ブレーキ
78             digitalWrite(motor_r2, LOW);
79             digitalWrite(motor_r1, LOW);
80             analogWrite(pwm_motor_r, 0);
81             break;
82     }
83 }
84 //左モータの動作

```

```

85 void run_Left(int n , int power){
86     switch(n){
87         case 0://後進
88             digitalWrite(motor_l1, LOW);
89             digitalWrite(motor_l2, HIGH);
90             analogWrite(pwm_motor_l,power);
91             break;
92         case 1://前進
93             digitalWrite(motor_l1, HIGH);
94             digitalWrite(motor_l2, LOW);
95             analogWrite(pwm_motor_l,power);
96             break;
97         case 2://ブレーキ
98             digitalWrite(motor_l2, LOW);
99             digitalWrite(motor_l1, LOW);
100             analogWrite(pwm_motor_l, 0);
101             break;
102     }
103 }
104
105 void setup(){ //実行時に 1回のみ実行
106     //それぞれの motor に対するピンを出力ポートに設定
107     pinMode(motor_r1, OUTPUT);
108     pinMode(motor_r2, OUTPUT);
109     pinMode(motor_l1, OUTPUT);
110     pinMode(motor_l2, OUTPUT);
111     Serial.begin(115200);
112     Serial.println("start");
113 }
114
115 #define THRESHOLD 0*8
116 #define THRESHOLD2 800*8
117 //sampling 8 times.
118 int scansensor(void){
119     static int stat=0;
120     ADCSRA&=0xFC;
121     ADCSRA |= 0x04;
122     int r=0,l=0,r2=0,l2=0;
123     for(int i=0;i<32;i++){
124         r+=analogRead(sensor_r)/4;
125         l+=analogRead(sensor_l)/4;
126         r2+=analogRead(sensor_r2)/4;
127         l2+=analogRead(sensor_l2)/4;
128     }
129     // Serial.print(r);
130     // Serial.print(",");
131     // Serial.print(r2);
132     // Serial.print(",");
133     // Serial.print(l);
134     // Serial.print(",");
135     // Serial.println(l2);
136
137     if(stat==0&&r<THRESHOLD&&r2<THRESHOLD&&l<THRESHOLD&&l2<
        THRESHOLD){
138         stat =1;
139     }
140     if(stat==1&&(r>THRESHOLD2||l>THRESHOLD2)){
141         stat=0;
142     }
143     if(stat==1){

```



```

144     return 0x8FFF;
145 }
146 return ((r-1)+2*(r2-l2))/8;
147 }
148
149 void loop(){ //制御プログラム
150     //センサから値を読み込む
151     while(1){
152         val_switch = analogRead(sensor_st);
153         // Serial.print("waiting\n");
154         // Serial.print(val_switch);
155         if (val_switch < 300){
156             while(analogRead(sensor_st)<600);
157             break;
158         }
159
160         run_Right(1,0);
161         run_Left(1,0);
162         digitalWrite(LED,LOW);
163     }
164     run_Right(1,255);
165     run_Left(1,255);
166     delay(1000);
167     while(1){
168         digitalWrite(LED,HIGH);
169         // val1 = (analogRead(sensor_r)+analogRead(sensor_r)+analogRead(sensor_r)+
170             // analogRead(sensor_r))/4;
171         // val2 = (analogRead(sensor_l)+analogRead(sensor_l)+analogRead(sensor_l)+
172             // analogRead(sensor_l))/4;
173         // //val3 = analogRead(sensor_r2);
174         // //val4 = analogRead(sensor_l2);
175         val_c = scansensor();
176         if(val_c==0x8FFF){
177             run_Right(0,AVERAGE);
178             run_Left(0,AVERAGE);
179         }else{
180             //読み取った値を PID 制御する
181             power_pid = pid_control(val_c);
182             //シリアルボードに制御した値を表示させる
183             Serial.print(val_c/8);
184             Serial.print(",");
185             Serial.println(power_pid);
186             //モータを左右でそれぞれ実行する
187             power_r = balance_r*(85 + power_pid)*AVERAGE/85;
188             power_l = balance_l*(85 - power_pid)*AVERAGE/85;
189             if (power_r > border){
190                 power_r = border;
191             }
192             if (power_l > border){
193                 power_l = border;
194             }
195             run_Right(1,power_r);
196             run_Left(1,power_l);
197         }
198     }
199     //delay(10);
200     while(millis()%10==0);
201     while(millis()%10!=0);
202     val_switch = analogRead(sensor_st);
203     if(val_switch < 300){

```

```
202     while(analogRead(sensor_st)<600);
203     break;
204   }
205 }
206 }
```

---

## 編集長より

### レポートについて

レポートは $\text{\LaTeX}$ を使って作成され、Git<sup>1</sup>を使って管理された。 $\text{\LaTeX}$ の特徴は、テキストベースのタイプセットができ、強力なレイアウト指定、数式のレンダリングが特徴である。

$\text{Word}^{TM}$ などのワードプロセッサとは異なりテキストベースであることで、常にレンダリング結果を意識することなく、原稿に集中して書くことができる。

また、レポートはGitを使って管理された。Gitの特徴の一つに、分岐させたブランチをまた合流(マージ)することができる。それぞれの作業内容は、ブランチを切って行い、その作業を最後に一緒にさせるという考え方だ。また、 $\text{\LaTeX}$ のビルドを通すのは意外と大変だ。一度エラーが出ると、 $\text{\LaTeX}$ はわかりにくいエラーを出す。そのためなるべく早いうちにエラーを見つけ、エラーを含んだソースコードを合流しないようにすることも必要だ。また、それぞれの人の作業内容は、一定期間ごとにmaster<sup>2</sup>にマージされていく。

### 自動ビルド

合流する前にビルドが通るか手動で確認するのは大変だ。そのため、GitLab<sup>3</sup>の機能であるコンテナの機能であるRunner<sup>4</sup>を実行する。ビルドが成功すると、ブランチにチェックが入り、成果物を確認してからマージすることができる。

## 編集量の調査

これは、12/28までの記録で、なおかつコミットの行数だけで図や整備などの手間は入っていないし、人によっては $\text{\LaTeX}$ の改行をうまく使わない人もいるので公平ではないことを断りたい。

このときは、レポートは41枚、ほぼライントレースの部分が完了しているところだ。

作業量は、かなりばらつきがあるように一見見えるが、編集長はそれぞれの編集を連結したり、移動する分、削除と追加が当然多くなる。PMの藤井は、プロジェクトの図を中心にかなり多く作業してくれた。そのため、文章をかなり多く書くということより図や資

---

<sup>1</sup>ソースコードなどの変更履歴の記録、追跡をするための分散型バージョン管理ツール

<sup>2</sup>メインのブランチ、分岐

<sup>3</sup>Gitのリモートリポジトリ

<sup>4</sup>この機能は、ある種の仮想化技術のよなもので、コンテナの中で小さな環境構築ができる。この中で $\text{\TeX}$ のビルドを自動化するコードを入れた

```

* commit 2cd7c7d72e5fd0dd1368de6b6d9225c03550e20 (HEAD -> master, origin/master, origin/HEAD)
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Sat Dec 28 23:48:05 2019 +0900

    学籍番号書いて〜

* commit cb5226db78450924239817eed45c5d8e1260c437
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Sat Dec 28 23:41:31 2019 +0900

    日誌を追加、勝手に編集長よりを追加

* commit c9c56b95fb112af04392375e6c129a0afe793800
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Sat Dec 28 23:07:15 2019 +0900

    Add Arduino's external file source code

* commit c6ae92259d6dfbc2fc23089f762ae4fa95ec9e9
Merge: 1897200 572b5de
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Sat Dec 28 23:05:22 2019 +0900

    Merge remote-tracking branch 'origin/master'

* commit 572b5de96fc242fa30b29a144297c7e981c99c23
Author: 仁美江原 <hitomi.cat0628@gmail.com>
Date: Sat Dec 28 03:47:00 2019 +0900

    学籍番号追加

* commit 2fcd025e04221e33772a884037f27d933786dff1
Merge: b876025 205660c
Author: 仁美江原 <hitomi.cat0628@gmail.com>
Date: Sat Dec 28 03:34:49 2019 +0900

    Merge branch 'master' of https://gitlab.com/team_nine1/team_nine_report

* commit 205660c1f5088dd961366115fcbf94aaf3dd441
Merge: 8a90166 147ebbc
Author: Fujii <>
Date: Thu Dec 26 16:33:26 2019 +0900

    Merge branch 'fujii'

* commit 147ebbc7b0be308c0f7990bff6820d1e231c34bb (origin/fujii)
Author: Fujii <>
Date: Thu Dec 26 14:43:28 2019 +0900

    図修正

* commit b876025bf3c82bd3c2a8c37f2194dcfd2e1a3f04
Author: 仁美江原 <hitomi.cat0628@gmail.com>
Date: Sat Dec 28 03:33:21 2019 +0900

    チーム報告1003-1017までまとめてみた

* commit 1897200a88cccd5cb2bf9670899ca8ffaabd216
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Sat Dec 28 22:56:07 2019 +0900

    図の配置を変更など

* commit f55bfff3f637ef68f00dbbfbcb43314572ee815d
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Thu Dec 26 20:33:53 2019 +0900

    delete extra section

* commit f39b354f842a2259f8ce0c17796f2e2771ba789
Merge: 8a90166 2501552
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Thu Dec 26 16:18:00 2019 +0900

    Merge remote-tracking branch 'origin/fujii'

* commit 250155284a85924cbe06a20a8a2872751b3bd78d
Author: Fujii <>
Date: Thu Dec 26 13:55:11 2019 +0900

    表の位置修正

* commit 8a90166de059c065cb02dd34e8da4387e0d8a81f
Merge: 2ceb7c6 9a63cad
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Thu Dec 26 15:25:37 2019 +0900

    Merge remote-tracking branch 'origin'

* commit 9a63cad7c1d0bb3d2a403f5984490930736babe6
Merge: ba25157 a34fb00
Author: 仁美江原 <hitomi.cat0628@gmail.com>
Date: Thu Dec 26 15:03:05 2019 +0900

    Merge branch 'master' of https://gitlab.com/team_nine1/team_nine_report

* commit ba2515766ec15a79e4103e8ae93fa98ad0e51d22
Author: 仁美江原 <hitomi.cat0628@gmail.com>
Date: Thu Dec 26 14:58:59 2019 +0900

    中級上級で機体画像を分離

* commit 2ceb7c6108b90160bfdb039657fb9b422f779084
Author: elect-gombe <13552050@elect-gombe@users.noreply.github.com>
Date: Thu Dec 26 15:23:50 2019 +0900

    Use bib

* commit a34fb004f2f5258bf52dd82418c4d01865431b9a (sawada)
Author: Sawada Yosuke <sawada.yosuke@chiba-u.jp>

```

図 2.17: Git のログの例

料の作成を頑張ってくれた。江原さんや沢田さんにあまり書くことを多く用意できなかったことは、次の追従のときには均等な作業量になるように直したいと思う。

表 2.6: 12/28 日現在のレポート編集行数

名前	追加	削除	編集行数の合計
大石	+1218	-685	1903
田中	+572	-78	650
藤井	+240	-88	328
澤田	+229	-20	229
江原	+269	-32	301

ともあれ，行数では語れないそれぞれの苦勞があるので，一概には言えないことを改めて申し上げます．

## 第3章 追従ロボット

### 3.1 追跡ロボットとは

#### 3.1.1 追跡ロボットについて

追跡ロボットとは、超音波センサー等で特定の物体の位置を検知し、その物体を追いかけるように走行するロボットのことである。自身と相手の距離を計測し、それに応じて機体のステアリング制御を行う。

#### 3.1.2 大会概要

ライントレースしながら走行する機体とそれを追従する機体を用いて、追従できた走行距離や走行時間について競う。

追従機体には、フォトリフレクタを取り付けてはいけない。追従するために、超音波センサーを用いる。

先導機体には、別途配布するアクリル板（20cm × 5cm）を取り付けてよい。このアクリル板は、追従機体の超音波センサーによる距離計測を容易にするために用いることを想定している。

走行時間は、追従機体がタイム計測器を通りすぎてから、先導機体が一周回って戻ってくるまでの時間とする。

最初の通過地点を通過したときに 10 点、2 番目の通過地点を通過したときに 20 点、1 周回ったときに 30 点が与えられる。通過地点あるいは先導機体のスタート地点にある細いラインを追従機体が完全に超えたときに通過したと判定する。

1 周走行時間に応じて、得点  $p$  が次式によって与えられる。

$$p = 50t_1/t$$

追従機体が先導機体に一度も衝突することなく、1 周回ることができたときに 20 点与えられる。

### 3.2 ハードウェアの構成

#### 3.2.1 回路図

ライントレーサの回路図を図 3.1 に示す。

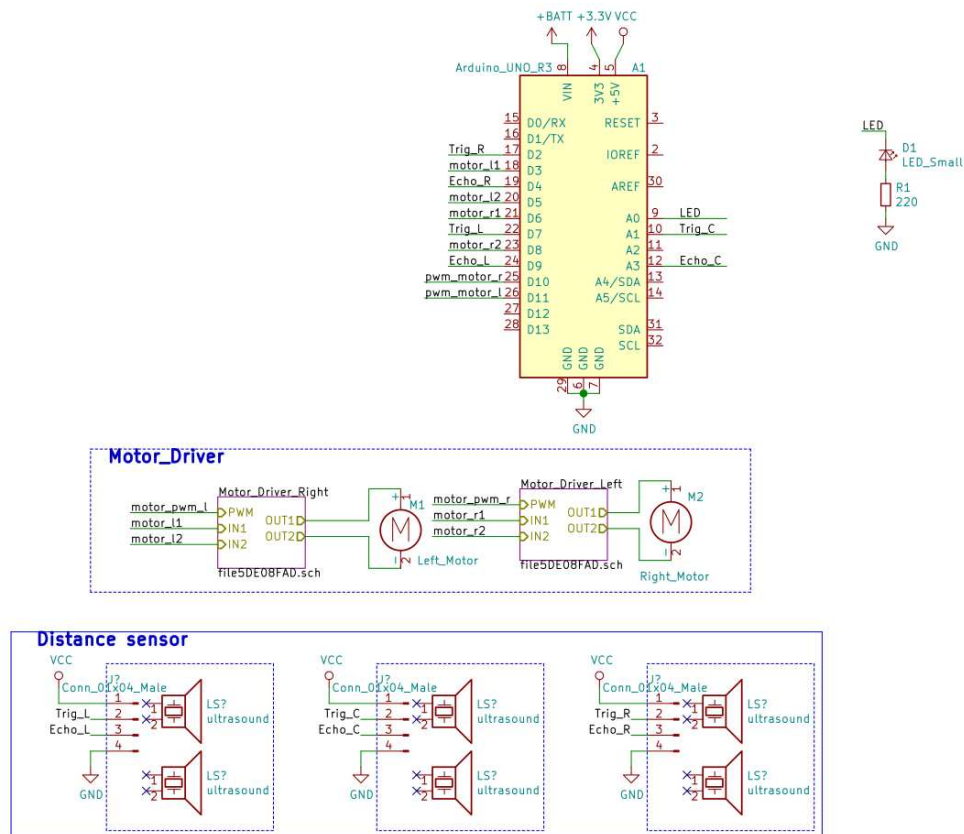


図 3.1: ライントレーサの全体回路図

## 3.3 メカニカル

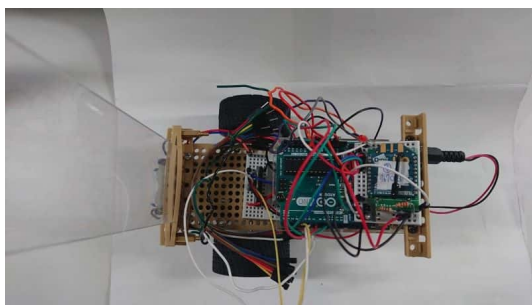
### 3.3.1 機体の構成

#### 先導機体について

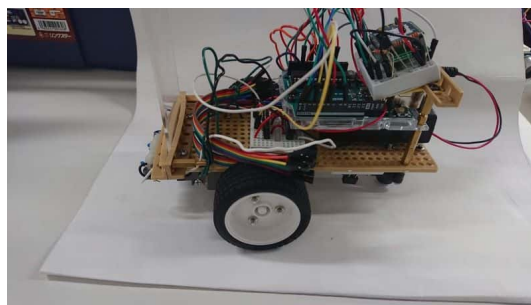
図 3.2 において、図 3.2(a) に先導機体を上から見た場合、図 3.2(b) に横から見た場合、図 3.2(c) に下から見た場合を示す。先導機体は、章 2.3 のラインレース中級用の機体を改造して制作した。主な変更点は、センサー部のフォトリフレクタを 2 つから 3 つに増やしたこと、機体後方にアクリル板を縦にして取り付けたこと、そして arduino の向きを逆にしたことである。フォトリフレクタを 3 つに増やしたことで、急カーブでも滑らかに走行することが可能になった。アクリル板を縦にしたことで追従機体が追従しやすくなったが、電池ボックスの arduino 接続端子部とアクリル板が接触してしまい故障する恐れがあったので、arduino の向きを逆向きにした。

#### 上級用の機体について

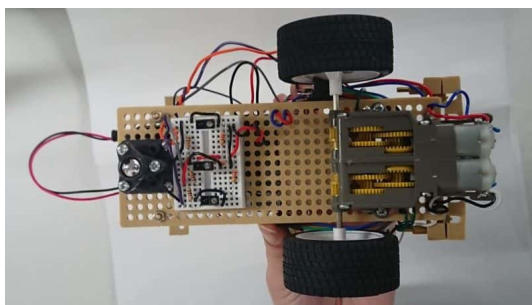
図 3.3 において、図 3.3(a) に上級用の機体を上から見た場合、図 3.3(b) に横から見た場合を示す。追従機体は、章 2.3 のラインレース上級用の機体を改造して制作した。主な



(a) 先導機体-上から



(b) 先導機体-横から



(c) 先導機体-下から

図 3.2: 先導機体の画像

変更点は、センサー部のフォトリフレクタ及びスタートボタン用のフォトリフレクタを全て取り除いたこと、機体前方に超音波センサーを放射線状に3つ取り付け付けたこと、そして機体後方にスタートボタン用のタッチセンサーを図 3.3(c) の様に取り付けたことである。追従機体にはフォトリフレクタを用いずに作成することが追従大会においてのルール1つだったため、フォトリフレクタは全て取り除き、スタートボタン用にタッチセンサーを取り付けた。超音波センサーを放射線状に配置したことで、先導機体を見失うことなく走行することを得意としている。

## 3.4 超音波センサについて

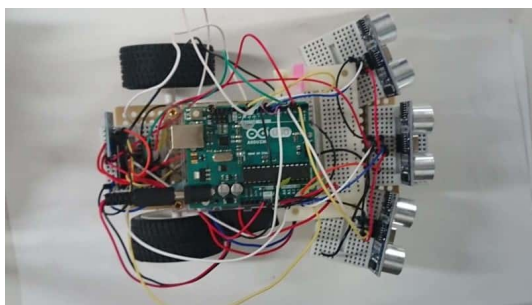
### 3.4.1 超音波センサーについて

#### 超音波センサー

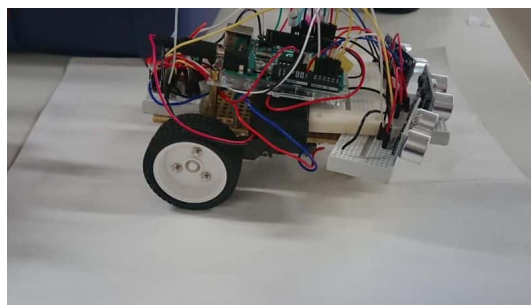
超音波は、直進性が強く、壁に当たると鋭く反射することで知られている。このため、距離を測るのに向いている。超音波は音の往復の時間を計測することで距離を測る。このため、精度が高く、比較的安定することで知られている。図 3.4 を見てほしい。光のように反射するため、扱うときは壁がスピーカーと水平、測る距離に対して垂直、またはある程度その方向に向いている必要がある。

そのため、これだけ距離が離れていて、平行からずれていれば当然光は帰ってこない。また、図 3.5 を見てほしい。センサー2つでは、機体のどこに先導するロボットがいるかどうかは実質上把握できない。

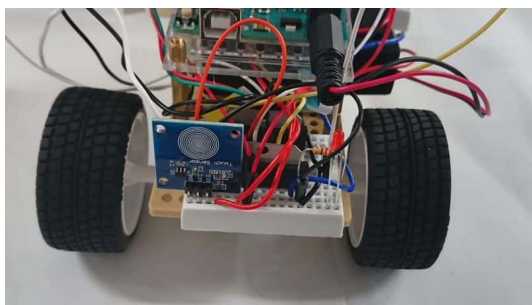
追従するためには、外れた時のスキャンニングが一つのカギになる。



(a) 追従機体-上から



(b) 追従機体-横から



(c) 追従機体-後ろから

図 3.3: 追従機体の画像

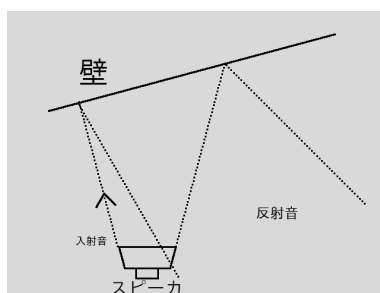


図 3.4: スピーカーの反射

Arduino で測るための簡易的なコードをおいておく．音が帰らないこともあるため，タイムアウトはきちんと設定しないと 1[s] も制御不能になる．

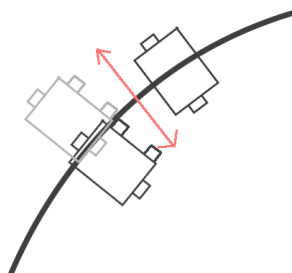


図 3.5: 2 センサーの例



---

```

1  #include <SimpleDHT.h>
2  #include <LiquidCrystal.h>
3
4
5  const int TRG = 5;
6  const int ECH = 4;
7  const int HIGHTIME=10;
8
9  int pinDHT11 = 2;
10 SimpleDHT11 dht11(pinDHT11);
11 LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
12
13 void setup() {
14     // put your setup code here, to run once:
15     pinMode(TRG,OUTPUT);
16     pinMode(ECH,INPUT);
17     Serial.begin(9600);
18     lcd.begin(16, 2);
19     lcd.print("Hello, _World!");
20 }
21
22 double GetTemp(void)
23 {
24     unsigned int wADC;
25     double t;
26
27     // The internal temperature has to be used
28     // with the internal reference of 1.1V.
29     // Channel 8 can not be selected with
30     // the analogRead function yet.
31
32     // Set the internal reference and mux.
33     ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
34     ADCSRA |= _BV(ADEN); // enable the ADC
35
36     delay(20); // wait for voltages to become stable.
37
38     ADCSRA |= _BV(ADSC); // Start the ADC
39
40     // Detect end-of-conversion
41     while (bit_is_set(ADCSRA,ADSC));
42
43     // Reading register "ADCW" takes care of how to read ADCL and ADCH.
44     wADC = ADCW;
45
46     // The offset of 324.31 could be wrong. It is just an indication.
47     t = (wADC - 324.31 ) / 1.22;
48
49     // The returned temperature is in degrees Celsius.
50     return (t);
51 }
52
53 void loop() {
54     byte temperature = 0;
55     byte humidity = 0;
56     int err = SimpleDHTErrSuccess;
57     if ((err = dht11.read(&temperature, &humidity, NULL)) != SimpleDHTErrSuccess)
58     {
59         Serial.print("Read _DHT11 _failed, _err="); Serial.println(err);delay(1000);
60         return;
61     }
62 }

```

```

60 }
61 double diff,dis;
62 // put your main code here, to run repeatedly:
63 noInterrupts();
64 digitalWrite(TRG, HIGH);
65 delayMicroseconds(HIGHTIME);
66 digitalWrite(TRG, LOW);
67 diff = pulseIn(ECH, HIGH,50000);
68 interrupts();
69 dis = (float)diff * 0.01715;
70 lcd.setCursor(0, 1);
71 lcd.print(dis);
72 lcd.print("cm");
73 lcd.print(temperature);
74 lcd.print("\xdf");
75 lcd.print("C░░░░");
76
77 Serial.print(dis);
78 Serial.print(",");
79 Serial.print(",");
80 Serial.println(GetTemp());
81 delay(1200);
82 }

```

---

## 距離の推定

距離の推定方法はいくつかある。

1. 点推定距離がわからない場合はとりあえず以前に測れたときの距離を返す。これが一番簡単だが、PID 制御とかしてる場合はかなり D 制御に影響を及ぼすから注意が必要。
2. 線で推定 2 回前から 1 回前の距離の差を 1 回前の測定分に足して推定する。

---

```

1 if(距離が測れなかった){
2   dist=一回前の距離+(一回前の距離-二回前の距離);
3 }
4 二回前の距離=一回前の距離;
5 一回前の距離=dist;
6 return dist;

```

---

## 3.5 ソフトウェア

### 3.5.1 ソフトウェアの構成

まず,3.4 の内容から,超音波センサは対象物との距離をはかるセンサであることが分かっている。その特性を踏まえた上で,対象物を追従するために,複数のセンサを用い,対処物(今回の実験では板)となす角度及び,中心との距離をはかり,角度と距離をそれぞれ制御することにした。

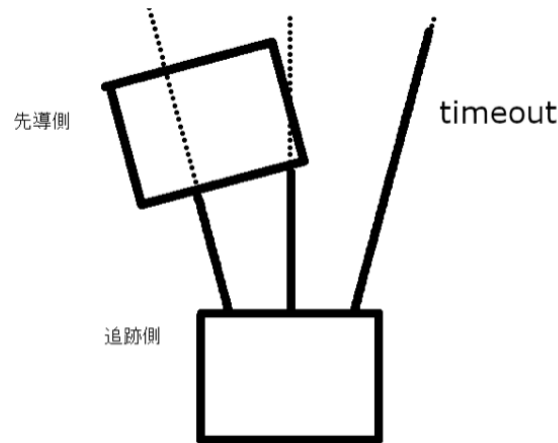


図 3.6: 検出方法 1

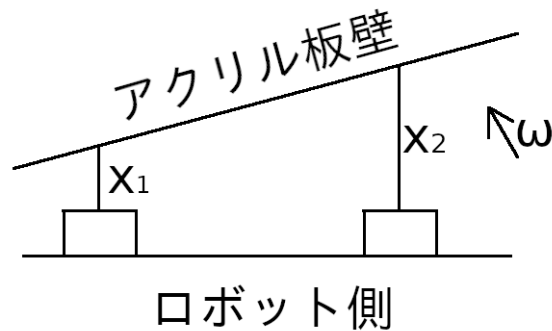


図 3.7: 検出方法 2

### 3.5.2 対象の検出

板との角度をはかるにあたって, 主な検出方法を二つ考えた.

一つ目は, センサを円周上に配置し, それぞれのセンサに重みをつけることで対象物となる角度を計測する方法である. この方法を図示すると, 以下のようになる. もう一つの検出方法として, 二つのセンサを水平に配置し, それぞれのセンサで計測した値から, 傾きを計測すること方法だ. 図示すると, 以下のようになる.

以上二つの検出方法のメリットとデメリットをまとめてみた.

前者 (図 3.6) の (以降, 周囲スキャン方式と呼ぶ) 方法では, 周囲をスキャンするために首振りが多くなるが, 対象物を見失いにくくなる. また, 十分なセンサ数がなければそもそも計測できず, 重みの設定などソースコードは複雑化する.

逆に, 後者 (図 3.7) の検出方法では, 対象となる板との正確な角度が計測できるが, 板との並行な成分でのズレを検出できず, 対象物を正面に捉えることができず急カーブなどでズレた場合にセンサの軌道上から外れる恐れがある. 周囲スキャン方式に比べ, 直線距離・角度を正確に計測できるため, 制御がしやすい. 以上のデメリット・メリットを踏まえ, 今回我々の班が選択したのは, 周囲スキャン方式である.

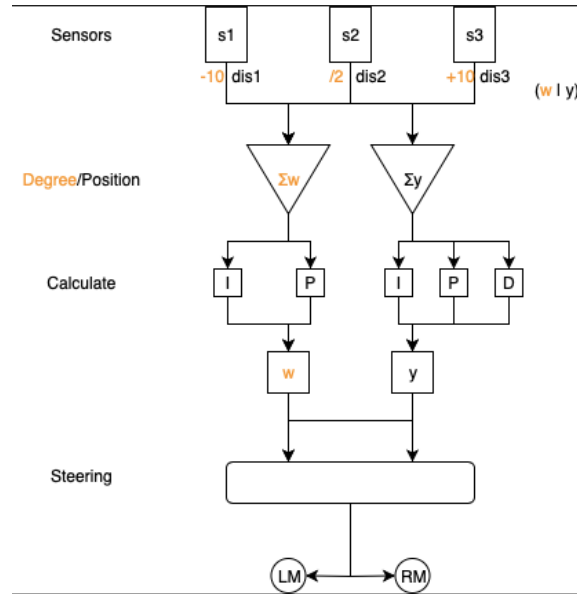


図 3.8: 簡易制御図

今回のコースが急カーブが多く、後者の検出方法では追従が難しいと判断した。また、距離の計測値については、各センサの内、最も距離の近いものをととする。

### 3.5.3 2 輪差動駆動

距離と角度の計測方法を決定した次に、それぞれの出力方法を定める。ロボット運動学にて基本である、2 輪作動駆動を用いた。前に進む成分を  $y$ , 旋回成分を  $\omega$  で分けると、モータへの出力  $(m_1, m_2)$  は以下ようになる。

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} = \begin{pmatrix} y \\ \omega \end{pmatrix}$$

以上から、左右のモータの出力  $m_1, m_2$  は、

$$m_1 = y + \omega$$

$$m_2 = y - \omega$$

のようになる。また、角度と距離の制御方法は、前回同様 pid 制御を用いた。以上、制御を簡単に図で表すと図 3.8 で表せる。

### 3.5.4 混線の除去

複数の超音波センサで距離をはかる上で、試走中に一つの問題点を発見した。それはあるセンサが発した音波を、別のセンサが拾ってしまうことだ。わかりやすく図で説明しよう。

図 3.9 において超音波センサの数は我々 9 班と同じ 3 個で、かつわかりやすい様に全て並

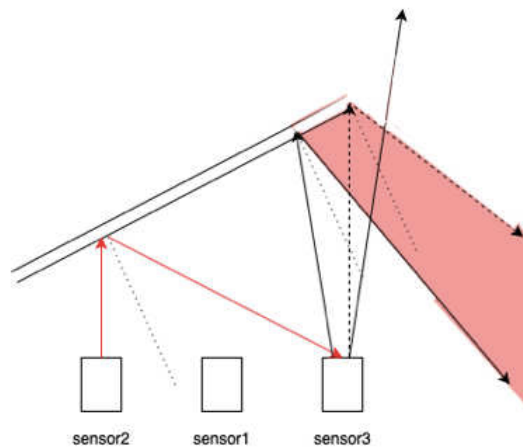


図 3.9: 音波反射図

行に置かれた状況でシミュレーションする。ここで、反射板が図の様に置かれたとき、右側に配置しているセンサ3は2つの黒矢印の間の範囲に音波を発するため、反射板で反射された音波の起動は赤で色付けした範囲となる。この範囲上にセンサ3は存在しないため、本来ならば測定値はなしとなるはずである。ところが、図の様に、左側のセンサ2から跳ね返った音波がセンサ3に返ってきてしまう場合が発生する。すると、先ほど理論上センサ3の測定値は0になるが、実際は0でない測定値が返ってきてしまうのだ。

そこで、この問題点を解決するために、超音波センサが値を読み取る時に、センサそれぞれで取得する間隔を空けることにした。超音波センサの商品ページのPDFデータシート[3]の図3.10を引用すると、今回実験で用いた超音波センサは、8パルス×40kHzの信号波で、これは間隔の非常に短いもので、全体で0.0002秒の長さしかない。よって、この間隔の短いパルスの集合を単純に一つの波とみなして考えることができる。このとき、センサ1つ目が作動した後に、別のセンサが作動するまでに十分な間隔を開けたとする。そうすることで、秒速約340m/sの音波はであり、空けた間隔分進み、他のセンサが作動する頃には波が通り過ぎている(図3.11)か、十分に弱まっている(図3.12)。これにより、センサが順に作動し、他のセンサが作動中に干渉しなくなり、正確な測定値を返す様になる。そもそも我々の班はセンサが放射状に配置されているため、元から干渉しづらくなっているが、この対策をしたことで、より正確性の高いセンサ値取得を行うことができる。

## 3.6 先導機体

ライントレースの上級コースと違い、今回の追従大会に用いられるコースの難易度が少し低めに設定されている。カーブ数が少なめで、かつ、急角度のカーブもないため、前回のライントレース中級コースで用いた機体を用いることにした。センサー2つの、PID制御で左右の出力を決定する、シンプルなソースコードをベースとしている。大会への調整として、スピードをあげ、PID各ゲインを変更した。ここで、さらなるスピードの底上げを行い、PID制御のみでは制御しきれない部分を、新たなセンサーを用いた脱線復帰処理機能をつけることで対応した。これは、前回のライントレース大会にて、他班が行っていた処理

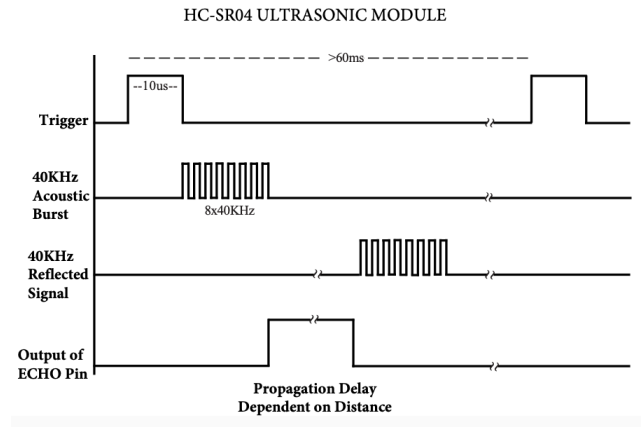


図 3.10: Module Timing 図

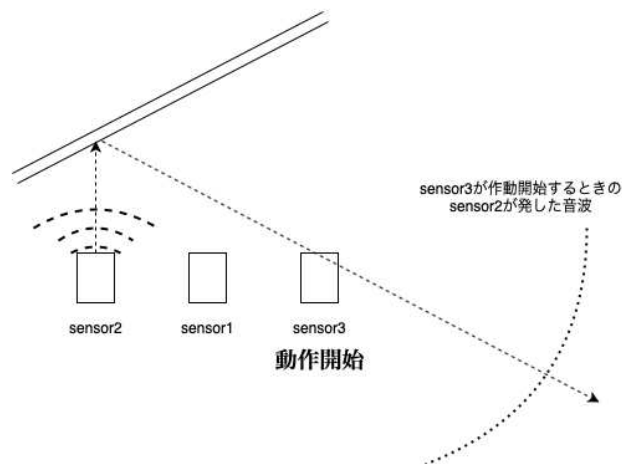


図 3.11: sensor 動作例 1

を参考にして追加した機能である。アルゴリズムとして、ハードの中心に取り付けたセンサーが白判定をした時には、読み取った偏差を更新しないというものだ。簡単に書くために、逆に考えて、黒判定のときのみ偏差を更新するという処理となる。上図 3.13, 今までの脱線前後での出力と復帰処理を施したあとの出力の比較図となる。

## 3.7 日誌

### 3.7.1 12月5日

#### 目標

1. 追従方針を立てる

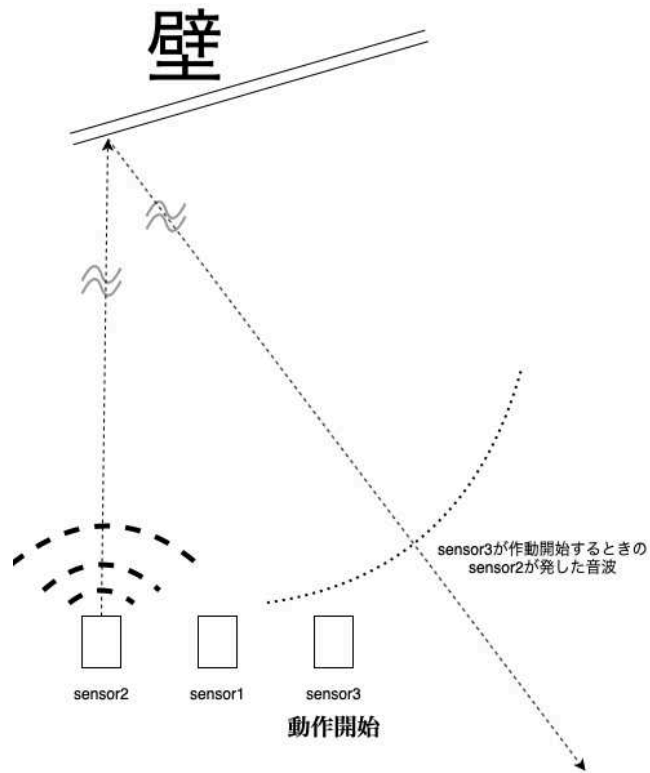


図 3.12: sensor 動作例 2

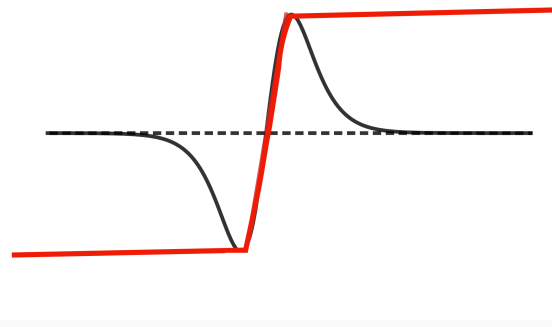


図 3.13: 復帰処理追加前後

2. 超音波センサーの特徴を掴む

3. 先頭マシンの仕様変更

## タスクの分担

チームのタスクは以下の3つである。

イ. 超音波センサーのテスト

ロ. ギア比, 先導機体の変更及び先導機体の走行テスト

ハ. 追従の方針の決定

個人で分担した作業は表 3.1 に示す.

表 3.1: 個人作業分担

追従マシンを高速ギアにする	藤井
追従のソフトウェアの作成	田中
先導マシンの調整	澤田 藤井 江原
追従のハードウェア・ソフトウェアの調整	大石

### タスクの達成度

イとハのタスクの達成度は 100% である. 超音波センサーの仕様を把握し, 追従の方針を決定することが出来た.

ロのタスクの達成度は 92% である. ギア比の変更は出来た. ライントレースの中級用の機体为先導機体として走らせることにしたので, そのための調整を行った. 追従大会のコースを走行することは可能になったが, 走行中に少し振動してしまった.

### 今後のスケジュール

次週以降は, 先導機体の調整を継続する. また実際に追従を行うことで課題点を洗い出し修正する.

## 3.7.2 12月12日

### 目標

- 先導車がコースを 1 周できるようにする
- レポート作成の目処を立てる.
- 追従機体を完成させ, 先導車を追従できるようにする.

### タスクの分担

個人で分担した作業を 3.2 に示す. また, 当初は先導機体の PID 制御を澤田と江原のみで行う予定だったが, 想定よりも時間がかかったため田中と大石に交代してタスクを完了した.



表 3.2: 個人作業分担

先導車に反射板の取り付け	藤井 澤田 江原
追従機体へのスイッチ接続	田中
追従機体の調整	大石

#### タスクの達成度評価

- 先導車に反射板の取り付け
- 追従機体へのスイッチ接続
- 追従機体の調整
- 先導機体の PID 制御

全てのタスクを授業時間内に完了させることができた。先導機体の PID 制御については、目標としたラップタイムである 20 秒を記録することはできなかったが、本番コースを走行できるようになった。

#### 今後のスケジュール

今回は追従機体の調整に使えた時間が 1 時間程度だったため、次回は遅れを取り戻すために作業効率を上げて取り組む必要がある。田中、大石は作業時間をフルに使って追従機体の調整を行う。他のメンバーは作業報告やスケジュール決め、プレゼン資料・レポート作成といった作業を積極的に消化する。

### 3.7.3 12 月 19 日

#### 今回の目標

速くゴールする。20 秒で一周する。

#### チームの作業

- 1, 先導機体の調整 藤井, 大石, 田中
  - 2, 追従機体の調整 田中, 藤井, 大石
  - 3, レポートを進める。澤田, 江原, 藤井
  - 4, レポート用に GitLab の使い方調べる。田中
- レポートを書くための環境を整えたので 100 % である。

### 作業ごとの達成度

- 1, 安定化はしたが, スピードを速くできなかったので, 50%
- 2, 安定化はしたが, スピードを速くできなかったので, 50%
- 3, レポートを空いた時間を使って進めることが出来たが, 予想以上に環境設定に時間が取られ思い通りには進めなかったもので, 60%である.
- 4, レポートを書くための環境を整えたので 100 %である.

### チームのタスク

- 1, 先導機体のパラメータ調整 1 周 20 秒未満でクリア
- 2, 追従機体のパラメータ調整 1 周 20 秒未満でクリア
- 3, レポートを進める. 全員がブランチ切って作業できるようにし, 自分の分を進める.

### タスクごとの達成度

- 1, スピードを速くできなかったが安定したので 50 %
- 2, スピードを速くできなかったが安定したので 50 %
- 3, 全員がブランチ切って作業できるようになったので 100 %

### 次回以降の課題

今回は, プレ大会である. 次週以降は, プレ大会に向けて調整を行い, プレ大会後は, プレ大会の結果と他の班の走行を踏まえて, よりスピードを出すにはどうしたら良いか検討していく.

また, 年内にライントレースについてのレポートを完成させることを目標にしているので, 各々作業を進めていく.

## 3.7.4 12 月 26 日

### 今回の目標

- ・ プレ大会直前の先導, 追従の 2 つの機体の調整
- ・ タスク分割した各班員の実験最終レポート (途中) のマージの 2 点となる.

### チームの目標

具体的な内容として, 前回までに 2 つの機体の動作に特に問題がなく, 一応完走するに至った. 今回のプレ大会では, 早いラップで周るより, 前回と同様の安定した走りを実装することで, 結果を出すことに専念した. よって, 前回の走行と同様のソースコード, 機体を用いるのはもちろん, 充電電池の電池残量 (電力) などの環境も同様に調整する必要がある. また, 2 つ目の作業については, 各班員が今週までに作成した, ライントレース大会についてまでのレポートのマージとなる. マージの際に, レポートの細かな修正も行うものとする.

## タスク達成度

1つ目の調整が80%,2つ目のマージが80%ほどである.

機体の調整について, プレ大会直前には安定して走ることができた. しかし, 実際の走行時には, 追従機のタイム測定用の超音波センサの波を拾ってしまい, 走行に乱れが生じてしまった. また, 先導機体も一部不安定な動作が見られたので, 完璧な調整とは言えなかった. マージについては, 各自が持ち寄った tex ファイルを git を用いて無事, マージすることができた. 修正は時間の関係から軽く行った程度で, 細かなものについては次週以降おこなうものとする.

## チームの目標

追従動作の安定化及び高速化が一つ. そしてもう一つに, レポートの修正及び内容の追加. この二点に重きをおく.

### 3.7.5 1月9日

#### 目標

- ライントレースの部分のレポートを完成させる
- プレゼン資料のドラフトを完成させる
- 先導機体を安定してラップタイム 20 秒を保てる (途中で止まらない, コースアウトしない)
- 追従が安定して追従できるようになる

#### タスクの分担

ライントレース部分のレポートに関して, pdf を連結するオートメーションの構築を大石  
ライントレースの部分のレポートのマージを大石

ライントレースの部分のレポートの日誌の集計を田中と澤田, ソフトウェアの解説が田中

プレゼン資料の作成の期待の紹介などが江原, 画像加工が澤田, 藤井が工夫した点, アルゴリズム, 仕組みの解説など

先導機体のフォトリフレクタ 1 つ追加し, ソフトウェアの調節が大石

追従機体のパラメータ調節が大石と田中のペア・プログラミング

## タスクの達成度評価

- ライントレースの部分のレポートに関して、pdfを連結するオートメーションの構築
- ライントレースの部分のレポートのマージ
- ライントレースの部分のレポートのチームの日志まとめ、ソフトウェアの解説

完全に完成、pdf連結のtaskをオートメーションのパイプライン処理で自動的にされるようになった。

- プレゼン資料の作成（江原，澤田，藤井）

⇒プレゼン資料の大まかなところは完了した。ただ、まだ練習までできていないので、発表の個人練習とスライドを完成させるところまでを来週までにして行けるところまでできたから 85 %

先導機体のセンサー追加（大石）は 100 %，追加して読み取りテスト，動作確認までできた。

先導機体の調節...95% まだ首振りが止まらないことがある。ただ，基本的にすべての動作は安定してできるようになり，コースアウトは一回もしなくなった。

追従機体：90 %の完成度まで落ちた。これは先導機体が速すぎてついていけないからで，また，高速なためたまに衝突したりロストすることがある。大会はたぶん乗り切れると思う。

### 3.7.6 1月16日

#### 今回の目標

- プレ大会を超える。
- プレゼンを完成,発表練習を成功する。

#### チームの作業

- 1, プレゼン資料の作成 藤井, 大石, 江原
- 2, プレゼンテーションの練習 藤井, 大石, 江原
- 3, 追従のハードウェア, 回路のレポート作成 大石
- 4, ソフトウェアのレポート作成 田中
- 5, テクニカルな部分のレポート作成 澤田
- 6, メカニカルな部分のレポート作成 江原

表 3.3: 本大会の結果

班	班名	通過得点	タイム (s)	タイム得点	衝突得点	チーム得点
9	ないん	30	15.86	50	20	100

#### 作業ごとの達成度

- 1, 資料は完成したが手直しがあるので 90%
- 2, 追従大会終了後に練習することができた.100%
- 3, 回路図を完成させた.100%
- 4, 追従機体について書き終えた.100%
- 5, 超音波センサーについて書けたがまだ push していないので 80%
- 6, メカニカルな部分について書けたがまだ push していないので 80%

#### チームのタスク

- 1, レポート作成 ソフト・ハード・技術・メカニカルのレポート
- 2, プレゼン資料作成 完成したプレゼン資料
- 3, プレゼン発表練習 発表の準備ができている状態

#### タスクごとの達成度

- 1, 書き終えてもまだ push していない部分が残っているので,90%
- 2, 資料は完成したが手直しがあるので 90%
- 3, 追従大会終了後に練習することができた.100%

#### 次回以降の課題

次回はプレゼンテーション大会である.

独自性をアピールできるような発表をしたい. 他の班のレポートの参考になったり, 他の班をレポートの参考に出来るようにプレゼンテーション大会に臨みたい.

## 3.8 追従大会

### 3.8.1 プレ大会考察

当日, 記録用のセンサーの使用によって挙動が不安定になった. しかし, そもそも PID 制御がうまくいっていたことに加え, 本番でも調整できたことが勝因である.

### 3.8.2 本大会結果

表 3.3 より, 15.86[s] のタイムを記録し, 1 位を取ることができた.

### 3.8.3 本大会考察

本番での調整がうまくいったことでさらなるタイム向上ができた。超音波の特性をきちんと理解した上で、アルゴリズムを考えられたことが勝因と言える。また、他の班に比べて、実験の試行回数が多かったことも勝因と言える。また、先導機体にも、脱線した際にどちらの方向にずれたかの復帰処理を加えたことで、安定した先導が可能になった。

### 3.8.4 追従大会のまとめ

安定性を追求するために、先導機体のセンサー数を増やした方が良かったと考えている。追従機体は、与えられた条件のもとで最高のものが出来た。

## 3.9 今後の展望

このセクションでは、今回用意されたパーツで作れる中ではかなりいい出来だったと思うが、今後もし機会があったらやりたいことを述べる。

### 3.9.1 超音波センサの追加

現在は3つの超音波センサを使用して、先導機体の方角をセンシングしている。

このため、方向は中央のほか、やや右、右、やや左、左の5つまでセンシングできる。ただ、これでは不十分なことも多い。

なかでも、振動が収束しない問題や、まだまだレンジが狭い問題、見失いやすい特徴などはこれから修正すべき点だと思う。

これらは、センスする放射状のレイの増加で対応できると考えている。

### 3.9.2 SLAMの実装

超音波センサで取得したデータをもとにマップを作成し、それをもとに自己位置と先導機体の位置を推定することで、より滑らかで追従性の高い機体を作れると考えられる。

しかし、超音波のままでは、半減角15度程度なので、かなり広い範囲から拾ってしまう。レーザのようなもっと鋭角なセンサを使う必要があると考えられる。

## 3.10 まとめ

追従大会に向けて追従ロボットの制作、先導ロボットの改良を行った。

追従ロボットは、超音波の調査など、基本を大切にして設計を行ったことで正確な追従が可能になった。

先導ロボットも、改良を行うことでコースアウトする欠点が克服できた。

結果として追従大会で1位を取ることができた。

## ソースコード

ソースコード 3.1: 追従用ソースコード

---

```
1 //Arduino Uno のピン配置
2 //超音波センサ
3 const int TRG_c = A1;
4 const int ECH_c = A3;
5 const int TRG_r = 2;
6 const int ECH_r = 4;
7 const int TRG_l = 7;
8 const int ECH_l = 8;
9 //右モーター
10 const int motor_r1 = 6;//Arduino の 3 番ピン
11 const int motor_r2 = 9;
12 //左モーター
13 const int motor_l1 = 3;
14 const int motor_l2 = 5;
15 //pwm 用のピン
16 const int pwm_motor_r = 11;
17 const int pwm_motor_l = 10;
18 //スイッチセンサ
19 const int sensor_touch = A4;
20 //スタート判定 LED
21 const int sensor_led = A0;
22 //超音波センサ判定用センサ
23 //角度パラメータ
24 int angle = 0;
25 //各センサーの重み
26 int weight_c = 0;
27 int weight_r = 0;
28 int weight_l = 0;
29 //出力パラメータ
30 int power_y = 0;
31 int power_w = 0;
32 int power_r = 0;
33 int power_l = 0;
34 //比例ゲイン
35 #define KP_y 8
36 #define KP_w 5
37 //積分時間
38 #define KI_y 0.4
39 #define KI_w 0.1
40 //微分時間
41 #define KD_y 0.20
42 #define KD_w 0.0
43 //短小時間
44 #define DELTA_T 0.050
45
46 #define HIGHTIME 10
47 #define TIMEOUT 2000
48 #define TARGET_DIS 9
49 #define TARGET_ANG 0
50
51 void setup() {
52   Serial.begin(115200);
53   pinMode(TRG_c, OUTPUT);
54   pinMode(ECH_c, INPUT);
55   pinMode(TRG_r, OUTPUT);
56   pinMode(ECH_r, INPUT);
```

```

57  pinMode(TRG_L, OUTPUT);
58  pinMode(ECH_L, INPUT);
59  pinMode(sensor_touch, INPUT);
60  pinMode(sensor_led, OUTPUT);
61  }
62
63  //超音波センサから距離を読み取る
64  double echo(int triger , int echo){
65      int diff;//時間の差分
66      float dis;//距離
67      digitalWrite(triger,HIGH);
68      delayMicroseconds(HIGHTIME);
69      digitalWrite(triger,LOW);
70      diff = pulseIn(echo,HIGH,TIMEOUT);
71      dis = (float)diff * 0.01715;
72      //Serial.println(dis);
73      delay(8);
74      return dis;
75  }
76
77  //pid制御
78  double pid_control_y(int val){
79      double p,i,d,pid;
80      static signed long diff_y[2];
81      static double integral_area_y;
82      diff_y[0] = diff_y[1]; //前回の偏差の値の格納
83      diff_y[1] = val - TARGET_DIS; //目標値と実際の測定値の偏差
84      integral_area_y += (diff_y[0] + diff_y[1]) / 2.0 * DELTA_T; //積分
85      p = KP_y * diff_y[1];
86      i = KI_y * integral_area_y;
87      d = KD_y * (diff_y[1] - diff_y[0])/DELTA_T;
88      pid = constrain(p+i+d, -255, 255); //-255 <= pid <= 255
89      return pid;
90  }
91
92  double pid_control_w(int val){
93      double p,i,d,pid;
94      static signed long diff_w[2];
95      static double integral_area_w;
96      diff_w[0] = diff_w[1]; //前回の偏差の値の格納
97      diff_w[1] = val - TARGET_ANG; //目標値と実際の測定値の偏差
98      integral_area_w += (diff_w[0] + diff_w[1]) / 2.0 * DELTA_T; //積分
99      p = KP_w * diff_w[1];
100     i = KI_w * integral_area_w;
101     d = KD_w * (diff_w[1] - diff_w[0])/DELTA_T;
102     pid = constrain(p+i+d, -255, 255); //-255 <= pid <= 255
103     return pid;
104 }
105
106 //右モータの動作
107 void run_Right(int n , int power){
108     switch(n){
109         case 0://後進
110             digitalWrite(motor_r1, LOW);
111             digitalWrite(motor_r2, HIGH);
112             analogWrite(pwm_motor_r,power);
113             break;
114         case 1://前進
115             digitalWrite(motor_r1, HIGH);
116             digitalWrite(motor_r2, LOW);

```



```

117     analogWrite(pwm_motor_r,power);
118     break;
119     case 2://ブレーキ
120         digitalWrite(motor_r2, LOW);
121         digitalWrite(motor_r1, LOW);
122         analogWrite(pwm_motor_r, 0);
123         break;
124     }
125 }
126 //左モータの動作
127 void run_Left(int n , int power){
128     switch(n){
129         case 0://後進
130             digitalWrite(motor_l1, LOW);
131             digitalWrite(motor_l2, HIGH);
132             analogWrite(pwm_motor_l,power);
133             break;
134         case 1://前進
135             digitalWrite(motor_l1, HIGH);
136             digitalWrite(motor_l2, LOW);
137             analogWrite(pwm_motor_l,power);
138             break;
139         case 2://ブレーキ
140             digitalWrite(motor_l2, LOW);
141             digitalWrite(motor_l1, LOW);
142             analogWrite(pwm_motor_l, 0);
143             break;
144     }
145 }
146
147
148 int scan_distsensor(int &mindis){
149     int angle;
150     double dis_center = echo(TRG_c,ECH_c);
151     double dis_right = echo(TRG_r,ECH_r);
152     double dis_left = echo(TRG_l,ECH_l);
153     mindis=20;
154     angle=0;
155     if (dis_right > 0){
156         angle-=10;
157     }
158     if (dis_left > 0){
159         angle+=10;
160     }
161     if (dis_center > 0){
162         angle=angle/2;
163     }
164     if(dis_right!=0){
165         mindis=dis_right;
166     }
167     if(dis_center!=0&&mindis > dis_center){
168         mindis=dis_center;
169     }
170     if(dis_left!=0&&mindis > dis_left){
171         mindis=dis_left;
172     }
173     return angle;
174 }
175
176 void loop() {

```

```

177 //タッチセンサ判定
178 while(1){
179     digitalWrite(sensor_led, LOW);
180     int sensor_val = digitalRead(sensor_touch);
181     if(sensor_val == HIGH){
182         while(digitalRead(sensor_touch) == HIGH);
183         break;
184     }
185     run_Right(1,0);
186     run_Left(1,0);
187 }
188 while(1){
189     digitalWrite(sensor_led, HIGH);
190     int mindis;
191     angle=scan_distsensor(mindis);
192     Serial.println(mindis);
193     // Serial.print(",");
194     // Serial.println(angle);
195     //追従制御
196     power_y = -pid_control.y(mindis);
197     power_w = -pid_control.w(angle);
198     // Serial.print(power_y);
199     // Serial.print(",");
200     // Serial.println(power_w);
201     power_r = (power_y + power_w);
202     power_l = (power_y - power_w);
203     if(power_r>0){
204         run_Right(1,power_r);
205     }else{
206         run_Right(0,-power_r);
207     }
208     if(power_l>0){
209         run_Left(1,power_l);
210     }else{
211         run_Left(0,-power_l);
212     }
213     while(millis()%50==0);
214     while(millis()%50!=0);
215     int sensor_val = digitalRead(sensor_touch);
216     if(sensor_val == HIGH){
217         while(sensor_val==HIGH)sensor_val = digitalRead(sensor_touch);
218         break;
219     }
220 }
221 }

```

---

### ソースコード 3.2: 先導用ソースコード

---

```

1 //Arduino Uno のピン配置
2 //右モーター
3 const int motor_r1 = 3;//Arduino の 3 番ピン
4 const int motor_r2 = 5;
5 //左モーター
6 const int motor_l1 = 6;
7 const int motor_l2 = 9;
8 //フォトリフレクタ
9 const int sensor_r = A3;
10 const int sensor_l = A1;
11 //const int sensor_r2 = A4;
12 //const int sensor_l2 = A5;

```

```

13 const int sensor_st = A2;
14 const int sensor_c = A4;
15 //LED
16 const int LED = A0;
17 //センサから読み取った値 (analog)
18 int val1 = 0;
19 int val2 = 0;
20 int val3 = 0;
21 //int val4 = 0;
22 int val_switch = 0;
23
24 //左右のセンサの差分
25 int val_c = 0;
26 //PID 制御された値
27 double power_r = 0.0;
28 double power_l = 0.0;
29 double power_pid = 0.0;
30 //目標値
31 #define target_val 0.0
32 //比例ゲイン
33 #define KP 0.049
34 //積分時間
35 #define KI 0.000//13
36 //微分時間
37 #define KD 0.0018//07
38 //短小時間
39 #define DELTA_T 0.010
40 //直線速度
41 #define AVERAGE 90
42 #define DEFALTE_AVERAGE 90
43 #define balance_r 0.85
44 #define balance_l 1.0
45 #define abs_pid_control_range 180
46 #define ratio_pid_control 0.85
47 //PID 制御する関数
48 double pid_control(int val){
49     double p,i,d,pid;
50     static signed long diff[2];
51     static double integral_area;
52     diff[0] = diff[1]; //前回の偏差の値の格納
53     diff[1] = val - target_val; //目標値と実際の測定値の偏差
54     integral_area += (diff[0] + diff[1]) / 2.0 * DELTA_T; //積分
55     p = KP * diff[1];
56     i = KI * integral_area;
57     d = KD * (diff[1] - diff[0])/DELTA_T;
58     pid = constrain(p+i+d, -255, 255); //-255 <= pid <= 255
59     return pid;
60 }
61
62 //右モータの動作
63 void run_Right(int n , int power){
64     switch(n){
65         case 0://後進
66             digitalWrite(motor_r1, LOW);
67             analogWrite(motor_r2, power);
68             break;
69         case 1://前進
70             analogWrite(motor_r1, power);
71             digitalWrite(motor_r2, LOW);
72             break;

```

```

73     case 2://ブレーキ
74         analogWrite(motor_r1, 0);
75         digitalWrite(motor_r2, LOW);
76         break;
77     }
78 }
79 //左モータの動作
80 void run_Left(int n , int power){
81     switch(n){
82         case 0://後進
83             digitalWrite(motor_l1, LOW);
84             analogWrite(motor_l2, power);
85             break;
86         case 1://前進
87             analogWrite(motor_l1, power);
88             digitalWrite(motor_l2, LOW);
89             break;
90         case 2://ブレーキ
91             analogWrite(motor_l1, 0);
92             digitalWrite(motor_l2, LOW);
93             break;
94     }
95 }
96
97 void setup(){ //実行時に 1回のみ実行
98     //それぞれの motor に対するピンを出力ポートに設定
99     pinMode(motor_r1, OUTPUT);
100    pinMode(motor_r2, OUTPUT);
101    pinMode(motor_l1, OUTPUT);
102    pinMode(motor_l2, OUTPUT);
103    Serial.begin(115200);
104    Serial.println("start");
105 }
106
107 int aread(int pin){
108     unsigned int aval=0;
109     for(int i=0;i<4;i++){
110         aval += analogRead(pin);
111     }
112     return aval / 4;
113 }
114
115 void loop(){ //制御プログラム
116     //センサから値を読み込む
117     while(1){
118         val_switch = analogRead(sensor_st);
119         //Serial.print("waiting\n");
120         //Serial.print(val_switch);
121         if (val_switch < 130){
122             while(analogRead(sensor_st)<300);
123             break;
124         }
125
126         run_Right(1,0);
127         run_Left(1,0);
128         digitalWrite(LED,LOW);
129     }
130     while(1){
131         val1 = aread(sensor_r);
132         val2 = aread(sensor_l);

```

```

133     val3 = read(sensor_c);
134     //val3 = analogRead(sensor_r2);
135     //val4 = analogRead(sensor_l2);
136     if(val3>200){
137         val_c = val1 - val2; //+ 2*(val3-val4);
138     }
139     //読み取った値を PID 制御する
140     power_pid = pid_control(val_c);
141     //シリアルボードに制御した値を表示させる
142
143     Serial.println(power_pid);
144
145     //モータを左右でそれぞれ実行する
146     int vel = AVERAGE;
147     digitalWrite(LED,LOW);
148     if(abs(val_c)>400){
149         digitalWrite(LED,HIGH);
150         vel = AVERAGE*3/4;
151         power_pid *= 4./3;
152     }
153     if(abs(val_c) < 100 ){
154         digitalWrite(LED,HIGH);
155         vel = AVERAGE*5/4;
156     }
157     power_r = ratio_pid_control * balance_r*(vel + power_pid);
158     power_l = ratio_pid_control * balance_l*(vel - power_pid);
159
160     //Serial.print(",");
161     //Serial.print(power_r);
162     //Serial.print(",");
163     //Serial.println(power_l);
164     if(power_r>0)
165         run_Right(1,power_r);
166     else
167         run_Right(0,-power_r);
168     if(power_l>0)
169         run_Left(1,power_l);
170     else
171         run_Left(0,-power_l);
172     //delay(10);
173     while(millis()%10==0);
174     while(millis()%10!=0);
175     val_switch = analogRead(sensor_st);
176     if(val_switch < 130){
177         while(analogRead(sensor_st)<300);
178         break;
179     }
180 }
181 }

```

---

## プロジェクト要件の達成度

本実験において、チームないんは役割分担や各大会に向けたスケジュール管理を適切に実施し、プロジェクトの要求を達成できたと考える。その要因は、大石を中心に技術や情報の共有を頻繁に行ない、チーム全員が常に実験の内容を把握できるようになったことである。蚊帳の外にされるメンバーがいなかったことで、追従大会まで士気を保って作業を

続けることができた。ハードウェアの性質を理解していたことが大会での結果に繋がったと考えられる。バッテリーの残量によってモーター出力が大幅に変化してしまうため常にバッテリーを充電することで対策したり、モーターの出力が左右で大きく違うことからソフトウェアに左右の出力バランスを調整する処理を挟んだり、ハードウェアの癖を見抜くことで理想に近い動きを実現出来た。ハードの性質はチームメンバーの気付きから見つかることもあったが、多くは大石の経験に基づくものであった。

プレゼン大会では、藤井と江原が構成を考えてからチーム全員でチェックしてスライドを完成させた。その際、チームの独自性が強調されるように話す順番を考えた上で、他のメンバーが論理性があるかどうかの判断を行なった。授業時間外に集まって練習をし、第1回大会では最も良い評価を得ることができた。

## 編集者より2

全体のレポートを一通りまとめたところで、編集時に気をつけたことに対して振り返りを行いたい。

今回のレポートのプロジェクトでは、 $\text{\LaTeX}$ を用いた原稿のタイプセットを行った。これに関しては、この規模のレポートをまとめるに十分な能力が確認できた。





















また、分散管理の手法としてデファクトスタンダードとなっているGitを用いた分散バージョン管理を用いた。学習コストが高いことで知られていて、実際に最後まで少し引きずりつつも無理やり使わせという感じだったが、結論からすると共同作業を強力にサポートしてくれた。

$\text{\LaTeX}$ のエラーは非常に読みにくい。自動ビルドシステムを設定することでエラーが伝搬しないように早い段階でエラーを発見し、またエラーを素早く発見することができた(図3.14)。結果としては、全体のエラーの発生自体も少なく、例えば写真のトラッキング忘れなどもすぐに気づくことができるようになった。

コミット数は、合計で265コミットである。コミットグラフを図3.15に示す<sup>1</sup>。

---

<sup>1</sup>注目すべきなのはコミット数ではなく、コミットの頻度の方だと考えている。忙しい中でみんな時間を作って定期的にコミットを入れてくれた人もいる。また、electgombeは大石で、複数重なっている人は、gitの設定ミスでコミットの人が分身しているだけである。私はgithubを普段使っていて、面倒だったのでちゃんと切り分けていない

All 244 Pending 0 Running 0 Finished 244 Branches Tags					Run Pipeline	Clear Runner
Status	Pipeline	Triggerer	Commit	Stages		
passed	#113738508 latest		master → c6d7fd55 Merge branch 'yu'	✓	00:01:32 11 hours ago	
passed	#113496051 latest		master → c6d7fd55 Merge branch 'yu'	✓	00:01:34 1 day ago	
passed	#113495576		master → d851ba45 Fix Day of daily"	✓	00:01:34 1 day ago	
passed	#113495572 latest		yu → 9354f775 #11 tracer追加	✓	00:01:24 1 day ago	
passed	#113494076		master → a9132d50 Merge remote-tracking branc...	✓	00:01:32 1 day ago	
passed	#113492326		master → d24e8096 重複を修正	✓	00:01:30 1 day ago	
passed	#113492225		master → 95257d7f バグ修正	✓	00:01:27 1 day ago	
passed	#113492196 latest		fujii → 95257d7f バグ修正	✓	00:01:29 1 day ago	
passed	#113491728		yu → 62b3fe5e bib@misc変更	✓	00:01:30 1 day ago	
passed	#113491582 latest		oishi → 03ae28fc Add future outlook	✓	00:01:34 1 day ago	
failed	#113491324		master → 704e1142 ソースコード内に文字がめり込...	✗	00:01:13 1 day ago	
passed	#113490709		yu → 0d58fa5c #10 daily表変更&混線防止追加	✓	00:01:38 1 day ago	
failed	#113490493		fujii → 704e1142 ソースコード内に文字がめり込...	✗	00:01:11 1 day ago	
passed	#113489665		master → 3a297aa9 Merge branch 'master' of http...	✓	00:01:33 1 day ago	
passed	#113488934		master → 57aad302 Merge branch 'master' of http...	✓	00:01:23 1 day ago	
passed	#113488392		oishi → c7dc181f Add sawada's daily new lines	✓	00:01:30 1 day ago	
passed	#113488219		master → 347588e1 marge	✓	00:01:43 1 day ago	
passed	#113487768		fujii → 347588e1 marge	✓	00:01:34 1 day ago	
passed	#113483081		master → 72ec9ad3 Merge branch 'master' of http...	✓	00:01:34 1 day ago	
passed	#113482557 latest		sawad → 193eff00 Merge branch 'master' into 'sa...	✓	00:01:28 1 day ago	

< Prev 1 2 3 4 5 ... 13 Next >

図 3.14: 自動ビルドの設定とそれぞれのビルドログ

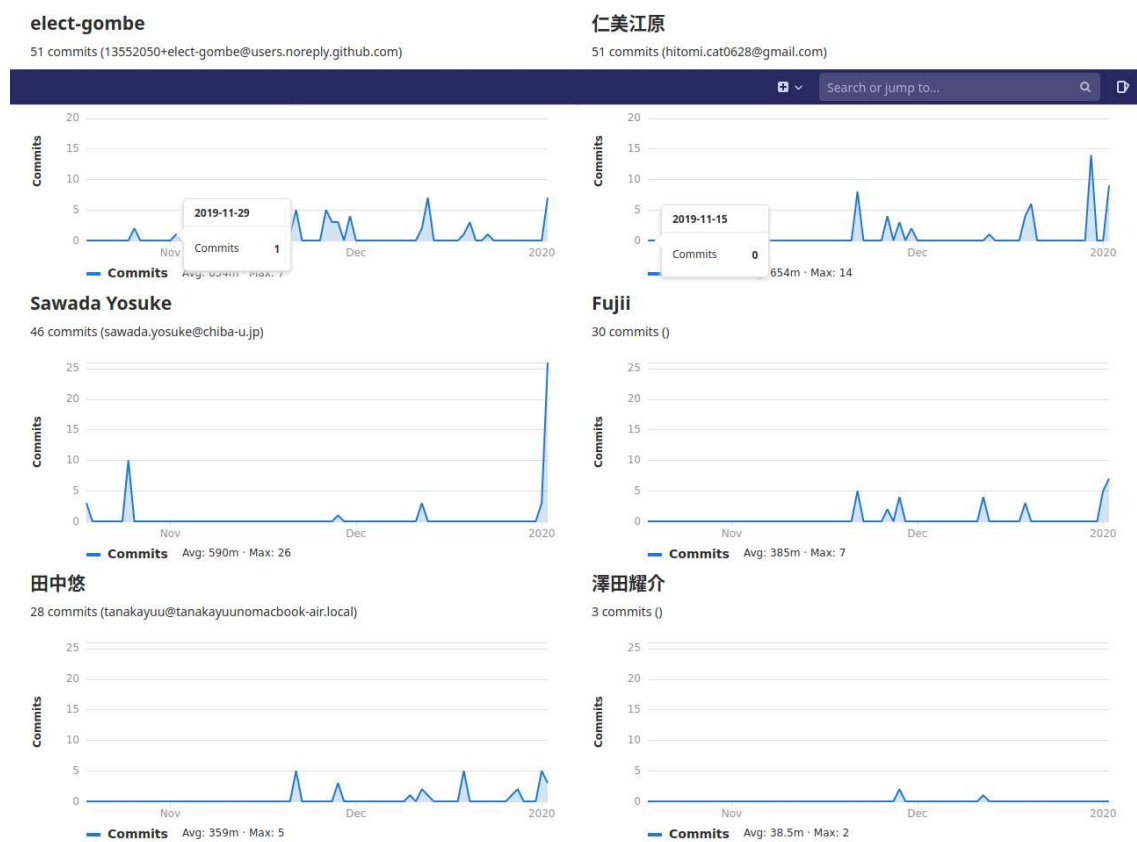


図 3.15: コミット数とアクティビティグラフ



## 実験全体のまとめ

ライントレーサの実験を通じてプロジェクトの管理，ハードウェアに関する基本知識，共同開発フローや発表について学ぶことができた。

プロジェクトの管理では，タスクの分担や評価，スケジュールの構築を通じて基本的なプロジェクト管理に関して学べた。

ハードウェアに関しては，ハードウェアの設計からソフトウェアとの強調に関しての基本的知識が学べた。

共同開発では，共同開発で起こりうる問題が勉強でき，バージョン管理ツールを用いた分散バージョン管理である Git の使い方を始め，基本的な知識が学べた。

プレゼンテーションでは，全体に自分たちの成果を発表するが，自分の班の特徴を最大限伝えられるようにするため効果的なプレゼンテーションをすることができた。特に班員でレビューを重ねることで，プレゼンテーションの問題点や修正点が発見でき，より良いものになったと確信している。

## 意見

もっと使用部品には柔軟性があればいいと思った。

たとえば，与えられた部品だけというのでは工夫点が出しにくい。創意工夫を出すために，もっといろいろな種類のパーツを使いたいと思った。具体的に困った点として超音波距離センサにはアクリル板を用いて追従するという点になっていた点がある。アクリル板を使う追従方法もあれば，他にも色々考えられる。それらに柔軟に対応させるためには反射素材の検討は重要になる。例えば画用紙やダンボール，電子部品以外にももっと様々な材料があればより多くのことを考えられるようになったと思う。

他にも，基板の大きさはもう少しあったほうが良かった。固定金具，例えばL字の固定具は欲しかった。

これらの材料は班によって必要不必要があると思うので，全班に配る必要はないと思うが，ルールで制約を設け必ずこうでなければならないということではなく，ほしいものを聞くなどの配慮は途中でもあっていいと思う。

## 参考文献

- [1] TOSHIBA. Ta7291p datasheet. <https://toshiba.semicon-storage.com/info/docget.jsp?did=16127&prodName=TA7291P>, 06 2007. (Accessed on 12/27/2019).
- [2] ELM-chan. Elm - 卓上ライントレースロボット. [http://elm-chan.org/works/ltc/report\\_j.html](http://elm-chan.org/works/ltc/report_j.html), 10 2003. (Accessed on 12/26/2019).
- [3] 秋月電子通商. Hc-sr04 pdf データシート. [http://akizukidenshi.com/download/ds/sainsmar/hc-sr04\\_ultrasonic\\_module\\_user\\_guidejohn\\_b.pdf](http://akizukidenshi.com/download/ds/sainsmar/hc-sr04_ultrasonic_module_user_guidejohn_b.pdf), 不明 不明. (Accessed on 01/30/2020).

## プロジェクト管理シート

プロジェクト管理に用いた Excel のシートのエクスポートした pdf を示す.

番号	カテゴリ	作業	責任者	Fujii	Tanaka
T1.1.1	調査	ライントレースの方法の調査	Fujii	0.5	0.5
T1.1.2	ハードウェア	ギアボックスの組み立て	Fujii	0.5	0.0
T1.1.3	ハードウェア	フォトトランジスタの回路作成	Ehara		
T1.1.4	ソフトウェア	Arduinoサンプルプログラムの動作確認	Sawada	0.5	0.5
T1.1.5	ハードウェア	モータードライバの回路作成	Tanaka		
T1.1.6	ハードウェア	モーターとモータードライバ、Arduinoの接続	Tanaka		0.5
T1.1.7	ソフトウェア	モータードライバの制御プログラム作成	Fujii	0.5	0.5
T1.1.8	ハードウェア	ハード全体の設計	Ohishi		
T1.1.9	ハードウェア	本体の組み立て	Ohishi		
T1.1.10	ソフトウェア	フォトトランジスタの値をプログラムから読み取る	Tanaka		0.5
T1.2.1	ハードウェア	ハード:フォトトランジスタとArduino接続	Ohishi	0.0	
T1.2.2	ハードウェア	ハード:モーターとArduino接続	Ohishi		
T1.2.3		ハード:ログ送信用のXBeeはんだ付け	Ohishi		
T1.2.4		ソフト:PID制御で求めた操作量をモーターに伝達する関数作成	Fujii	0.5	
T1.2.5		ソフト:PID制御の計算関数	Tanaka		1.0
T1.2.6		ソフト:Xbeeのドライバー等環境設定	Fujii	0.5	
T1.2.7		ソフト:Xbeeで2台のArduinoをペアリング	Ohishi	0.5	
T1.2.8		ソフト:シリアルモニターでXbeeの接続テスト	Ohishi		
T1.2.9		ソフト:PID制御のプログラムとモーター制御のプログラムをマージ	Fujii	0.5	0.5
T1.2.10		ハード:センサをトレーサー本体に接続	Ehara		
T1.2.11		ソフト:制御方法の話し合い	Ohishi	1.0	1.0
T1.3.1		ハード:モーターONOFFスイッチの接続	Sawada		0.5
T1.3.2		ハード:ブレッドボードの固定	Ehara		
T1.3.3		制御方法の確立と共有	Ohishi	0.5	0.5
T1.3.4		ソフト:シリアルモニタからパラメータを変更するプログラム	Fujii	3.0	
T1.3.5		ソフト:PID制御の調整	Tanaka		3.0
T1.3.6		ソフト:Xbeeのボーレート変更,ペアリング	Ohishi	0.5	
T1.4.1		ソフト:制御周期の安定化	Sawada		0.5
T1.4.2		ハード:機体Bのモーター位置変更	Ehara		
T1.4.3		ハード:機体Aのセンサー位置変更	Ohishi		
T1.4.4		ソフト:機体Bに制御プログラム導入	Fujii	0.5	
T1.4.5		ハード:機体Bの配線	Ehara		
T1.4.6		ソフト:機体AでPID制御の調整	Tanaka		2.0
T1.4.7		ハード:タイヤの組み立て	Fujii	1.0	
T1.4.8		ハード:機体Bのギヤ比率を低速に	Fujii	1.0	
T1.4.9		ソフト:2台のArduino通信	Ohishi		
T1.5.1		プレゼン資料作成	Fujii	2.0	
T1.5.2		ハード:機体Bの配線	Ehara		
T1.5.3		ハード:機体Aのメンテナンス	Ohishi		

T1.5.4	ソフト:PID制御の値調整	Tanaka		2.0
T1.5.5	プレ大会			
T1.6.1	ソフト:機体B(上級)の調整	Fujii		1.0
T1.6.2	レポート作成	Ehara		
T1.7.1	レポート分担作業	Ohishi		
T1.7.2	ソフト:中級マシンの調整	Fujii	1.0	
T1.7.3	ソフト:上級マシンの調整	Tanaka		
T1.7.4	レポート記述	Sawada	1.0	
T2.1.1	超音波追従マシンの技術選定	Ohishi		1.0
T2.2.1	ハード:追従機のギア変更	Fujii	0.5	
T2.2.2	技術選定	Ohishi	0.5	1.5
T2.2.3	追従機のソフトウェア記述	Tanaka		2.5
T2.2.4	先導機の調整	Sawada	1.5	
T2.2.5	追従機の配線	Ohishi	1.0	
T2.3.1	実験レポート計画	Ohishi	0.5	0.5
T2.3.2	先導車に反射板を取り付ける	Ehara	1.0	
T2.3.3	先導機体のPID制御	Sawada	1.5	2.0
T2.3.4	追従機体にスイッチ接続	Tanaka		0.5
T2.3.5	追従機体の調整	Ohishi		1.5
T2.4.1	先導機体の調整	Tanaka	1.0	1.0
T2.4.2	追従機体の調整	Ohishi	0.5	1.0
T2.4.3	レポートを進める	Sawada	1.5	
T2.4.4	レポート用にGitLabの使い方調べる	Tanaka		0.5
T2.4.5	レポート書くための環境設定	Tanaka		0.5
T2.5.1	ソフト:先導機体のPIDゲイン調整	Tanaka		1.0
T2.5.2	ソフト:追従機体の調整	Ohishi		1.0
T2.5.3	レポートのマージ	Ohishi		
T2.5.4	レポート作成	Ehara	3.0	1.0
T2.5.5	プレ大会			
T2.6.1	pdf連結のオートメーション構築	Ohishi		
T2.6.2	レポートのマージ	Ohishi		1.0
T2.6.3	チーム日誌まとめ	Tanaka		2.0
T2.6.4	プレゼン資料作成	Fujii	3.0	
T2.7.1	プレゼン資料作成	Fujii	3.0	
T2.7.2	ソフト:先導機体調整	Ohishi		
T2.7.3	追従大会			
T2.7.4	レポート作成	Sawada		3.0
集計			34.0	36.0

Sawada	Ehara	Ohishi	作業量合計	所要時間	
	0.5	0.5	0.5	2.5	0.5
	0.0	0.0	0.5	1.0	0.5
		0.5		0.5	0.5
	0.5			1.5	0.5
		0.5		0.5	0.5
	0.5	0.5		1.5	0.5
	0.5			1.5	0.5
			0.5	0.5	0.5
			0.5	0.5	0.5
	0.5			1.0	0.5
		0.5	0.5	1.0	0.5
		0.5	0.5	1.0	0.5
			0.5	0.5	0.5
				0.5	0.5
				1.0	1.0
				0.5	0.5
				0.5	0.5
		0.5		0.5	0.5
				1.0	0.5
		1.5		1.5	1.5
		0.5	1.0	3.5	1.0
	1.0		0.5	2.0	1.0
		0.5		0.5	0.5
	0.5	0.5	0.5	2.5	0.5
			0.5	3.5	3.0
	3.0	3.0	1.5	10.5	3.0
			1.0	1.5	1.0
	0.5			1.0	0.5
		1.0		1.0	1.0
			0.5	0.5	0.5
				0.5	0.5
		1.0		1.0	1.0
	2.0		2.0	6.0	2.0
				1.0	1.0
		0.5		1.5	1.0
			1.0	1.0	1.0
			0.5	2.5	2.0
		2.0		2.0	2.0
			0.5	0.5	0.5

2.0		1.0	5.0	2.0
			0.0	0.0
		1.0	2.0	1.0
1.0	1.0		2.0	1.0
0.5		0.5	1.0	0.5
1.0	1.0	0.0	3.0	1.0
		1.0	1.0	1.0
1.5			2.5	1.5
		1.0	2.0	1.0
			0.5	0.5
		1.0	3.0	1.5
	2.5	2.0	7.0	2.5
2.5			4.0	2.5
		0.5	1.5	1.0
0.5	0.5	0.5	2.5	0.5
1.5	1.5		4.0	1.5
1.5	2.0	1.5	8.5	2.0
		0.5	1.0	0.5
		1.0	2.5	1.5
		1.0	3.0	1.0
		1.0	2.5	1.0
2.5	3.0	1.0	8.0	3.0
0.5			1.0	0.5
			0.5	0.5
		1.0	2.0	1.0
		1.0	2.0	1.0
		1.0	1.0	1.0
3.0	3.0		10.0	3.0
			0.0	0.0
		1.0	1.0	1.0
		2.0	3.0	2.0
			2.0	2.0
3.0	3.0		9.0	3.0
	3.0	2.0	8.0	3.0
		1.0	1.0	1.0
			0.0	0.0
3.0			6.0	3.0
			0.0	0.0
			0.0	0.0
33.5	34.0	37.0	174.5	

(No. )

[illegible]



## 各作業の進捗状況

(No. )

[illegible]

## 各作業の進捗状況

(No. )

[illegible]

## 各作業の進捗状況

(No. )
