



チーム紹介、目標、意気込み

私達K-Labは、宮崎大学工学部情報システム工学科に所属している学部生を中心としたチームです！

今大会に向けての目標としては、「前大会の優勝チームを超える」です。そのために、カラーブロック中央4隅、駐車完了で最低でも26ポイント、通常走行26秒を目標としています。意気込みとしては、チーム開発を円滑に行うためにGitHubを用いました。また、チームメンバーの書いたソースコードに対して、上級生がコードレビューを行いました。GitHubで困ったことがあつたら私達に何でも聞いてください！！

モデルの概要

- 昨年優勝チームのタイムを更新することを目標とし、具体的な数値として総合走行時間0秒とした。
- テストコードの記述を容易にするため、モック化しやすい構造にしている。また、EV3RTのタイヤ制御やカラーセンサーなど、各センサー類の操作を容易にするための工夫をした。
- LコースとRコースで各エリアの処理をまとめた。これにより、どちらのコースを呼び出すかコンパイル時に決定可能である。
- ゲーム攻略において旋回制御と色識別は非常に大事である。そこで、旋回制御では、正確に角度を推定するためにノイズを考慮に入れた状態観測器を作成した。また、誤認識を軽減するため循環バッファを利用した色識別を採用した。

モデルの構成

1. 要求モデル

目標を設定し、ミスユースケース図を用いて要求を定義、要求図を用いて要求分析を行った。

2. 分析モデル

Rコースのゲーム攻略におけるオブジェクトをオブジェクト図およびクラス図で定義し、要素定義をモデル化した。また、走行体の動作定義として、走行体の制約を検討した。要素定義と動作定義を基に、攻略の指針を決定し、走行体に伴う制約と攻略の指針から、解法を導き出した。

3. 設計モデル

要求を満たすシステムを実現する構造をクラス図で定義した。定義したクラス図から、全体の処理の振舞をシーケンス図とステートマシン図で、ゲーム攻略の処理の振舞をシーケンス図で、それぞれ定義した。

4. 制御モデル

上位モデルで定義した各要求を満たす要素技術の内、重要な要素技術を記述した。

1-1. 目標 総合走行時間 0秒

開発目標は「昨年優勝チームのタイムを更新すること」とし、具体的な数値は以下のように算出した。

$$(目標走行タイム) - (目標ボーナスタイム) = (目標リザルタイム)
26.0秒 - (最低) 26.0秒 = 0.0秒$$

目標リザルタイムは、昨年の全国大会における優勝チームの点数が1.8秒であるため、1.8秒未満と設定した。



Rコースでは、黒線以外を通るルートで得られる利点は少ないので、Fig1-1-1の矢印のような経路が最短経路であると判定した。

走行体のPID制御における最高速度は約400mm/sであった。旋回時の最高速度をほぼ同値として確認できた。Fig1-1-1のように走行タイムエリアを走行すると、全体で計24.7秒で走行タイムエリア全体を走行可能であると算出した。

- ボーナスタイムを確実に獲得するための戦略
バワープロックがどの初期位置であった場合でも、中央4つにカラーブロックを有効移動した場合、バワースポット設置によるボーナスタイムが最低「8秒」確保できる。また、直角駐車場停止ボーナスの獲得を目指す。
- 目標ボーナスタイムの算出
ブロック並ベアリ亞では、ブロック有効移動で「13秒」とバワースポット設置で「8秒」、また直角駐車場エリ亞で「5秒」をボーナスタイム獲得目標とし、合わせて目標ボーナスタイムを「26秒」と設定した。



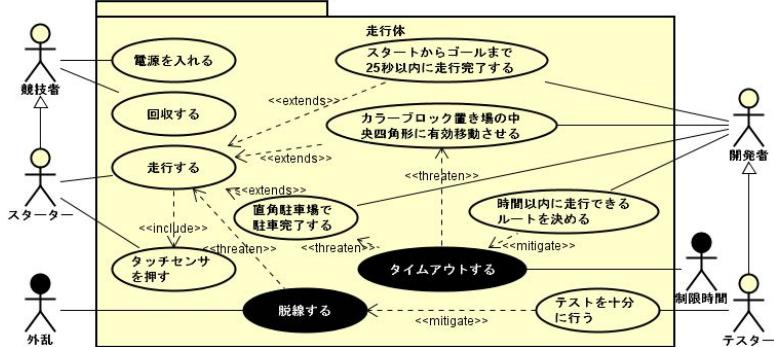
内側の白い部分とカラーブロックが一部でも重なっていれば良い。

項目	ボーナスタイム(秒)	目標タイム1案(秒)	目標タイム2案(秒)
ボーナス合計タイム(秒)	26	26	13
ブロック有効移動個数1個	5	0	0
ブロック有効移動個数2個	9	0	0
ブロック有効移動個数3個	11	0	0
ブロック有効移動個数4個	13	13	0
バワースポット設置1個あたり	2	8	8
直角移動駐車	5	5	5

1-2. 要求定義

1-2-1. ミスユースケース図

ユーザが走行体に求める機能をミスユースケース図で分析した。



1-2-2. ユースケース記述

代表的なユースケースのユースケース記述を用いてより詳細な分析を行った。

ユースケース名	U1_スタートからゴールまで25秒以内に走行完了する
目的	走行タイムを25秒以内にするため
アクター	開発者
不変条件	走行体の電源がONである
事前条件	走行体が正常に動作している
事後条件	走行体がスタートライン、中間ゲート1、中間ゲート2、ゴールゲートを通過している
包含	「タッチセンサを押す」ユースケース
基本フロー	S1. アクターが初期位置コードを入力する S2. アクターが初期位置を走行体を設定する include:タッチセンサを押す S3. 走行体はスタートからゴールまで走行する
代替フロー	-
例外フロー	-
ユースケース名	U4_走行テストを十分に行う
目的	プログラムの信頼性を確認するため
アクター	テスター
不変条件	開発者がプログラムを作成していること
事前条件	走行体の電源がONである
事後条件	-
基本フロー	S1. テスターは走行体が到達する目標位置を設定する S1-1. 走行体が到着する目標位置を設定する S1-2. スターターの行動を再現する S1-3. 走行体が目標位置まで到着することを確認する S1-4. S1-2とS1-3の目標位置を100%目標位置までの割合を確認する S2. 単体テストを行う
代替フロー	E1. S4で100%達成しなかった場合はプログラムの修正を行いS1に戻る
例外フロー	-
ユースケース名	U5_時刻以内に走行できるルートを決めらる
目的	「タイムアウトする」ミスユースケースを緩和するため
アクター	開発者
不変条件	-
事前条件	「直角駐車場で駐車完了する」ユースケース完了時に競技時間が2分を超えていない
事後条件	「直角駐車場で駐車完了する」ユースケース完了時に競技時間が2分を超えていない
基本フロー	S1. ゴールゲートからカラーブロック置き場の座標コード8に移動する S2. 青のカラーブロックを座標コード10に移動する S2-1. 青のカラーブロックを座標コード5に移動する S2-2. 黄のカラーブロックを座標コード6に移動する S2-3. 黄のカラーブロックを座標コード10Cに移動する S3. 座標コード10背面に直角駐車場で移動する
代替フロー	A1. A1とS1終了時に座標コード8にブロックが配置されている場合 A2. S1終了時に座標コード8にブロックが配置されている場合
例外フロー	-
戻り	T1. 移動するカラーブロックの配置が悪い T1-1. カラーブロックを置きたい場所に他のカラーブロックが配置されている T1-2. 最後に到達する座標コード8にブロックが配置されている T2. ブロックの配置手順が計算できない T3. パワープロックを移動させてしまう

Fig1-2-2 ユースケース記述

1-3. 要求分析

1-1節の目標分析と1-2節の要求定義を用いて要求分析を行った。要求図をFig1-3-1に示す。各要求は凡例のように分類した。上位目標と下位目標は、それぞれユースケース記述におけるユースケース名とその基本フローにおける各ステートメントに一致している。Fig1-3-1は、下位目標を更に分析するために記述した。

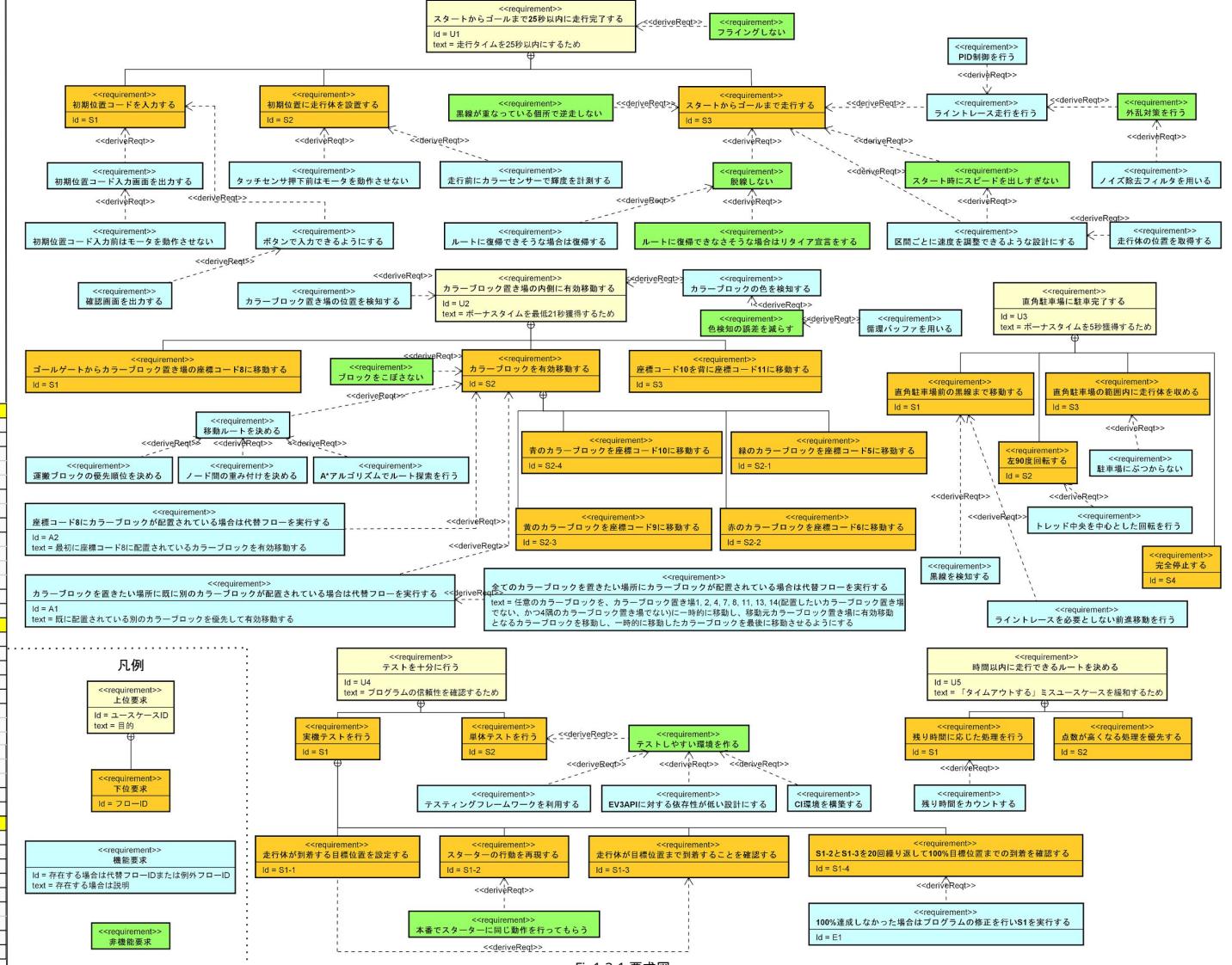
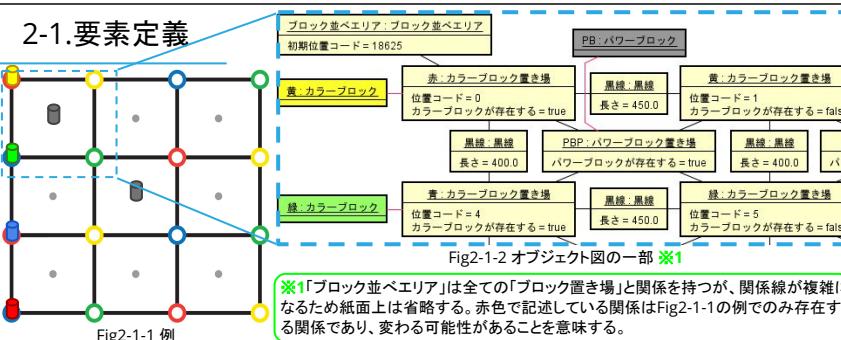


Fig1-3-1 要求図

2-1.要素定義



2-2.動作定義

2-2-1.ライントレース以外での移動方法の開発を無視

走行体がカラーブロック置き場間を移動する方法の1つにライントレースがある。しかし、移動ルートは黒線に沿う必要があるため、カラーブロック置き場から別のカラーブロック置き場へ移動する際に、黒線を用いて下を向いているため、カラーブロックの色を判定するためにはカラーセンサーを利用する。しかし、移動ルートは黒線に沿う必要があるため、カラーブロック置き場から別のカラーブロック置き場へ移動する際に、黒線を用いて下を向いているため、カラーブロックの色を判定するためにはカラーセンサーを利用する。

斜め移動が必要になるカラーブロックの初期位置をFig2-2-1に示す。走行体は位置コード8からスタートするため、位置コード4, 8, 9, 12にブロックが存在する場合、位置コード8のブロックを移動させることは困難である。したがってライントレース以外での移動方法が必要になる。

カラーブロックが赤青黄の順で配置されると仮定した場合、カラーブロックが置かれるカラーブロック置き場の色によって、組合せ数は変動する。例えば、カラーブロックが、緑、赤、黄のカラーブロック置き場の順で置かれた場合、組合せ数は $12 \times 12 \times 10 \times 10 = 14400$ となる。また、カラーブロックが、カラーブロック置き場の黄、赤、緑の順で置かれた場合、組合せ数は $12 \times 11 \times 10 \times 10 = 13200$ となる。したがって、組合せ数はカラーブロック置き場の色の順番の組合せ数 $3 \times 3 \times 3 = 27$ で場合分けできる。さらに、カラーブロック置き場の色の順番の組合せが、青赤黄、青青黄、青黄赤、黄青赤の場合に、Fig2-2-1の初期位置になる組合せが2つずつある。よって、それぞれの場合のFig2-2-1の初期位置になる確率を求める式は下式の通りである。

以上により、Fig2-2-1と同じ初期位置になる確率は極めて低いため、開発効率の観点からライントレース以外での移動方法はしないこととした。

Fig2-2-1 不可能な配置例

$$\text{青赤黄の場合} \dots \frac{2}{12 \times 11 \times 11 \times 10} = \frac{2}{14520}$$

$$\text{青黄赤の場合} \dots \frac{2}{12 \times 11 \times 11 \times 10} = \frac{2}{14520}$$

$$\text{青青黄の場合} \dots \frac{2}{12 \times 11 \times 12 \times 10} = \frac{2}{15840}$$

$$\text{黄青赤の場合} \dots \frac{2}{12 \times 11 \times 11 \times 10} = \frac{2}{14520}$$

$$\frac{2}{14520} + \frac{2}{15840} + \frac{2}{14520} + \frac{2}{14520} \times 100 = 0.001998\% (%)$$

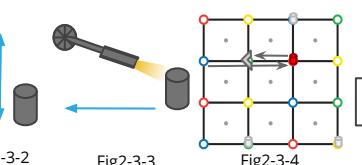
2-2-2.アームによる色判定時の挙動

カラーブロックの色を判定する際にカラーセンサーを利用する。カラーセンサーは通常時、ライントレースなどで下を向いているため、カラーブロックの色を判定するためにはカラーセンサーの回転軸とカラーセンサーの先端には数センチの距離が存在する(Fig2-3-2の青矢印参照)ため、カラーブロックの直前でアームの長さ分だけ後退してアームを上昇させなければならない(Fig2-3-3の青矢印参照)。

2-2-3.パックステップ

設置したカラーブロックに走行体が接触してしまい、有効移動となったカラーブロックが無効化してしまう恐れを防ぐため、走行体がカラーブロックをカラーブロック置き場に運搬したのちに、運搬経路探索結果における運搬経路リストの最後の値の1つ前の位置に後退する。

例えば位置コード4に赤のカラーブロックが存在した場合、位置コード4～5～6の順序で運搬する。その後、走行体は位置コード5に移動する(Fig2-3-4)。



2-2-4.ブロック運搬時の挙動

走行体がブロックを保持している状態でブロックエリアを移動する場合、ブロックはアーム内に固定されていないため、移動の際にブロックの取りこぼしを防ぐ走行が必要となる。

左折時(右折時):一定の速度以上にならなければ遠心力によるブロックの移動を最小限にすることが可能であるため、問題なく左折(右折)できる。



2-3.指針

開発目標で確実に中央の4つのブロック置き場にブロック有効移動を成立する事を目標として設定した。以降、中央四角形と呼ぶ。ブロックがどのような初期位置でも中央四角形を完成するため、初期位置(Fig2-1-2)から中央四角形までのブロックの運搬優先順序について検討する。

Fig2-3-5とFig2-3-6のような、ブロックXの運搬予定先に別のブロックYが存在し、そのブロックYの運搬予定先はブロックXの存在するブロック置き場であるような状況が発生する可能性がある。この状態を移動ループと呼ぶ。移動ループが判明するのは、最初のブロックのブロック置き場の色と最後のブロックの色が一致するか判定できる、ブロック色識別後である。Fig2-3-2、Fig2-3-3、Fig2-3-5、Fig2-3-6、Fig2-3-7の場合、ブロックを別の場所に一時的に避難する必要がある。

Fig2-3-8において、数字が小さいほど優先順位が高い。

最も優先順位が高いのは位置コード8である。これは、Fig2-3-1～Fig2-3-4において、運搬妨害ブロックとしても最も邪魔であると判断したためである。次に、運搬妨害ブロックが初期位置として配置される可能性がある中央四角形が、2番目の運搬優先順序として決定した。最後に、進路の妨げになりにくい4隅を最後の運搬優先順序として決定した。

同時に、位置コストと経路コストを定義した。位置コストとして、運搬優先順序の数値をそのまま流用する。また、黒線の経路コストを1(赤字)と決定した。例外としてカラーブロックが存在するカラーブロック置き場に接した黒線の経路コストは99とした。

Fig2-3-9 Fig2-3-8の経路が完全に塞がれている場合

Fig2-3-10 Fig2-3-9の経路が完全に塞がれている場合

Fig2-3-11 Fig2-3-10の経路が完全に塞がれている場合

Fig2-3-12 Fig2-3-11の経路が完全に塞がれている場合

Fig2-3-13 Fig2-3-12の経路が完全に塞がれている場合

Fig2-3-14 Fig2-3-13の経路が完全に塞がれている場合

Fig2-3-15 Fig2-3-14の経路が完全に塞がれている場合

Fig2-3-16 Fig2-3-15の経路が完全に塞がれている場合

Fig2-3-17 Fig2-3-16の経路が完全に塞がれている場合

Fig2-3-18 Fig2-3-17の経路が完全に塞がれている場合

Fig2-3-19 Fig2-3-18の経路が完全に塞がれている場合

凡例

カラーブロック

未確定カラーブロック

パワーブロック

2-4.解法

Fig2-3-1 位置コード8にブロックが存在する場合

指針を基に、ブロック並ベエリアにおける初期位置としてブロックが存在するブロック置き場に運搬優先順序を決定した。ブロック並ベエリアに運搬優先順序の番号を追記した画像を、Fig2-3-8に示す。Fig2-3-8において、数字が小さいほど優先順位が高い。

最も優先順位が高いのは位置コード8である。これは、Fig2-3-1～Fig2-3-4において、運搬妨害ブロックとしても最も邪魔であると判断したためである。次に、運搬妨害ブロックが初期位置として配置される可能性がある中央四角形が、2番目の運搬優先順序として決定した。最後に、進路の妨げになりにくい4隅を最後の運搬優先順序として決定した。

同時に、位置コストと経路コストを定義した。位置コストとして、運搬優先順序の数値をそのまま流用する。また、黒線の経路コストを1(赤字)と決定した。例外としてカラーブロックが存在するカラーブロック置き場に接した黒線の経路コストは99とした。

Fig2-3-10 Fig2-3-9の経路が完全に塞がれている場合

Fig2-3-11 Fig2-3-10の経路が完全に塞がれている場合

Fig2-3-12 Fig2-3-11の経路が完全に塞がれている場合

Fig2-3-13 Fig2-3-12の経路が完全に塞がれている場合

Fig2-3-14 Fig2-3-13の経路が完全に塞がれている場合

Fig2-3-15 Fig2-3-14の経路が完全に塞がれている場合

Fig2-3-16 Fig2-3-15の経路が完全に塞がれている場合

Fig2-3-17 Fig2-3-16の経路が完全に塞がれている場合

Fig2-3-18 Fig2-3-17の経路が完全に塞がれている場合

Fig2-3-19 Fig2-3-18の経路が完全に塞がれている場合

2-4-1.移動ルートのコスト計算について

移動ルートのコストはFig2-3-8より、位置コストと経路コストの和をスコアとして算出する。最小のスコアを持つ未確定カラーブロックが存在するカラーブロック置き場が複数ある場合、位置コードの値が小さいものを優先して処理する。例えば、位置コード2、4、13、15にカラーブロックが存在し、走行体の位置コードが8である場合(Fig2-4-1-I)、位置コード2のスコアは1×4+位置コスト3=7である。同様に、位置コード4のスコアは4、位置コード13のスコアは5、位置コード15のスコアは8である。よって、最小スコアの位置コード4のカラーブロック置き場に移動する(Fig2-4-1-II)。

2-4-2.運搬ルートのコスト計算について

運搬ルートのコストはFig2-4-9より、経路コストの和をスコアとして算出する。例えば、ブロック置き場の位置コード2、4、13、15にカラーブロックが存在し、走行体の位置コードが8である場合(Fig2-4-1-I)、位置コード2のスコアは1×4+位置コスト3=7である。同様に、位置コード4のスコアは4、位置コード13のスコアは5、位置コード15のスコアは8である。よって、最小スコアの位置コード4のカラーブロック置き場に移動する。

2-4-3.ブロック並ベエリア攻略中のステートマシン

ブロック並ベエリア攻略中の、走行体の状態遷移を表したステートマシン図を、Fig2-4-2に示す。2-2-1節より、最初は位置コード8に移動する。また、Fig1-3-1におけるU2「カラーブロック置き場の内側(中央四角形)に有効移動する」のS3「位置コード10を背に位置コード11に移動する」より、最後は位置コード11に移動する。

ルート探索中状態から次の状態に遷移する場合、次の状態は一時避難中状態と移動中状態と運搬中状態の3つの可能性が存在する。一時避難中状態には、条件A(Fig2-3-2に該当する状態)または条件B(Fig2-3-2に該当する状態)、または条件C(Fig2-3-5とFig2-3-6に該当する状態)または条件D(Fig2-3-7に該当する状態)の場合に遷移する。移動中状態と運搬中状態は、一時避難中状態に遷移しない場合に遷移する。移動中状態は、走行体の移動前位置にカラーブロックが存在しない場合、または運搬しようとしているカラーブロックの運搬予定先にブロックが存在する場合(Fig2-3-5およびFig2-3-6の前半部に該当する状態)に遷移する。後半の場合には、移動中状態に移る前にバックステップを実行する必要がある。

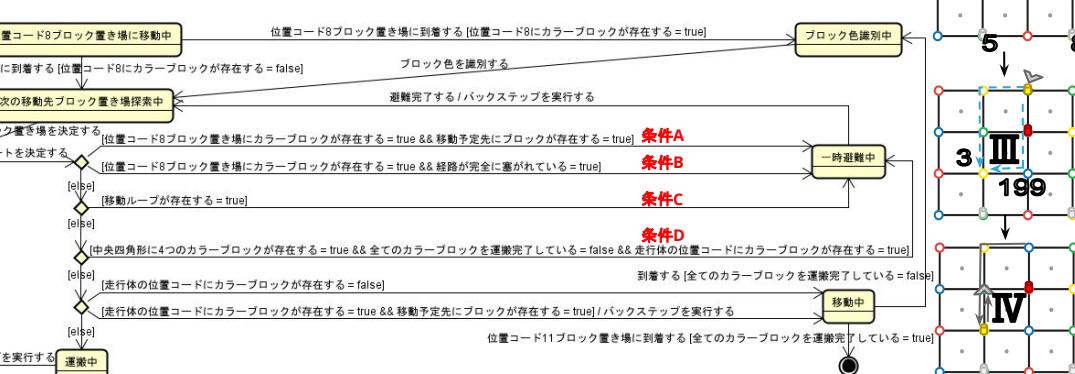


Fig2-4-1

3-1.構造の概略

要求を満たすシステムを実現するための検討をした。構造のパッケージ図、および各パッケージの分割基準を、Fig.3-1-1に示す。

Fig3-1-1を見ると、上位層のほうが依存関係が高く、下位層のほうが低くなっている。また、「ev3api」パッケージを「走行体」パッケージに含めている。その理由を、次に示す。

- 走行体が変わる場合はev3apiおよび下位層が大きく変わることがあり、下位層に置く必要性が低いため
 - 走行体パッケージをモック化することで、下位層のテストコードの記述が容易になるため

Rコースのエリア(走行エリア・ブロック並ベエリア・直角駐車エリア)の呼び出しを1クラスでまとめると、

Rコースオブジェクトが各エリアの「攻略する()」メソッドを呼び出すことで、各エリア間の関係を少なくでき、エリアをモックオブジェクトで差替えやすい。

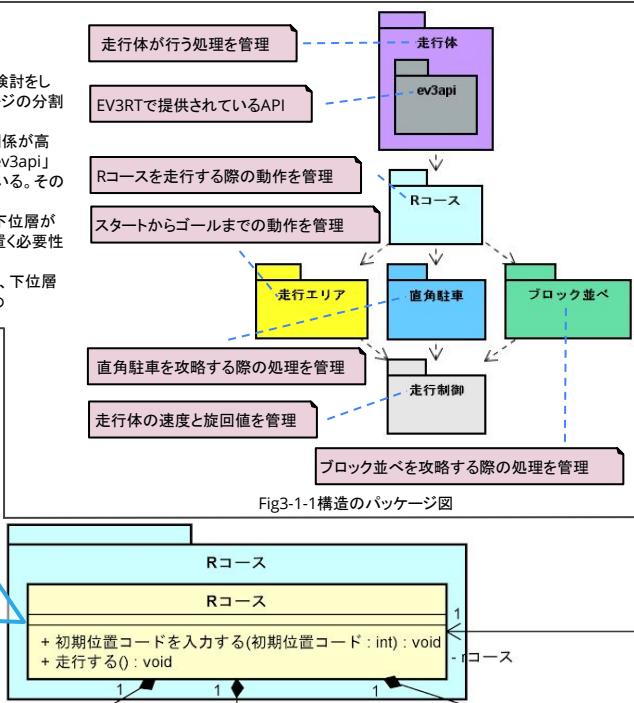


Fig3-1-1構造のパッケージ

3-2.構造の詳細

さらに詳細な構造分析を行った。煩雑さを軽減するため、getter・setter記を一部省略する。また、走行エリアと直角駐車は簡略する。

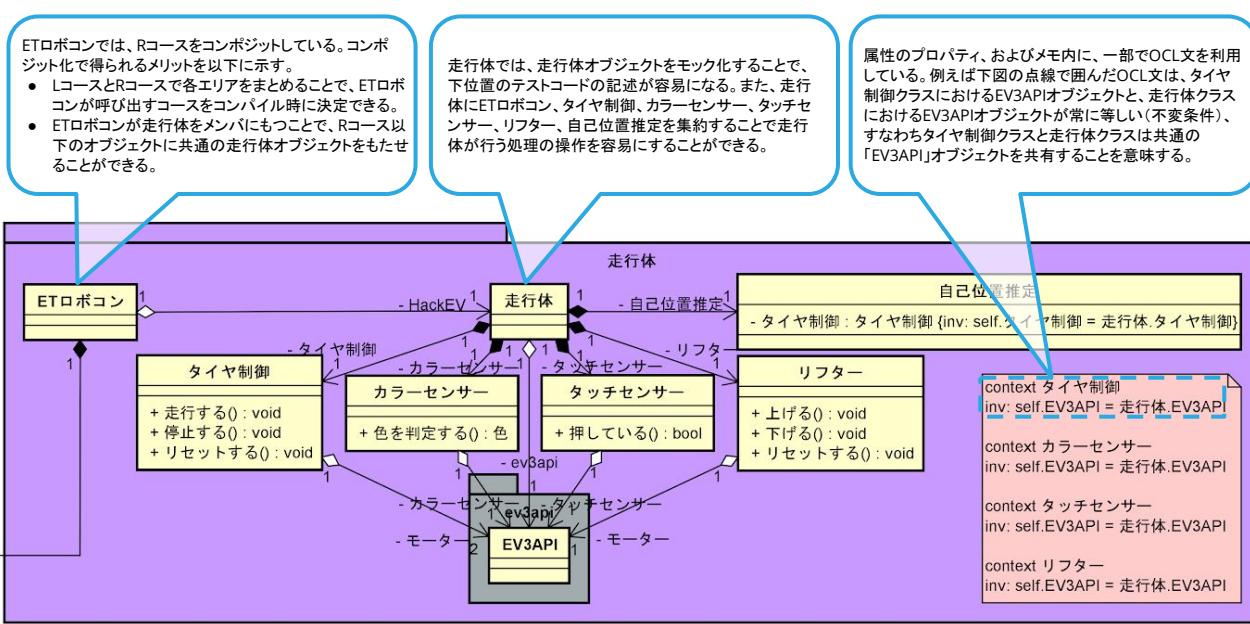
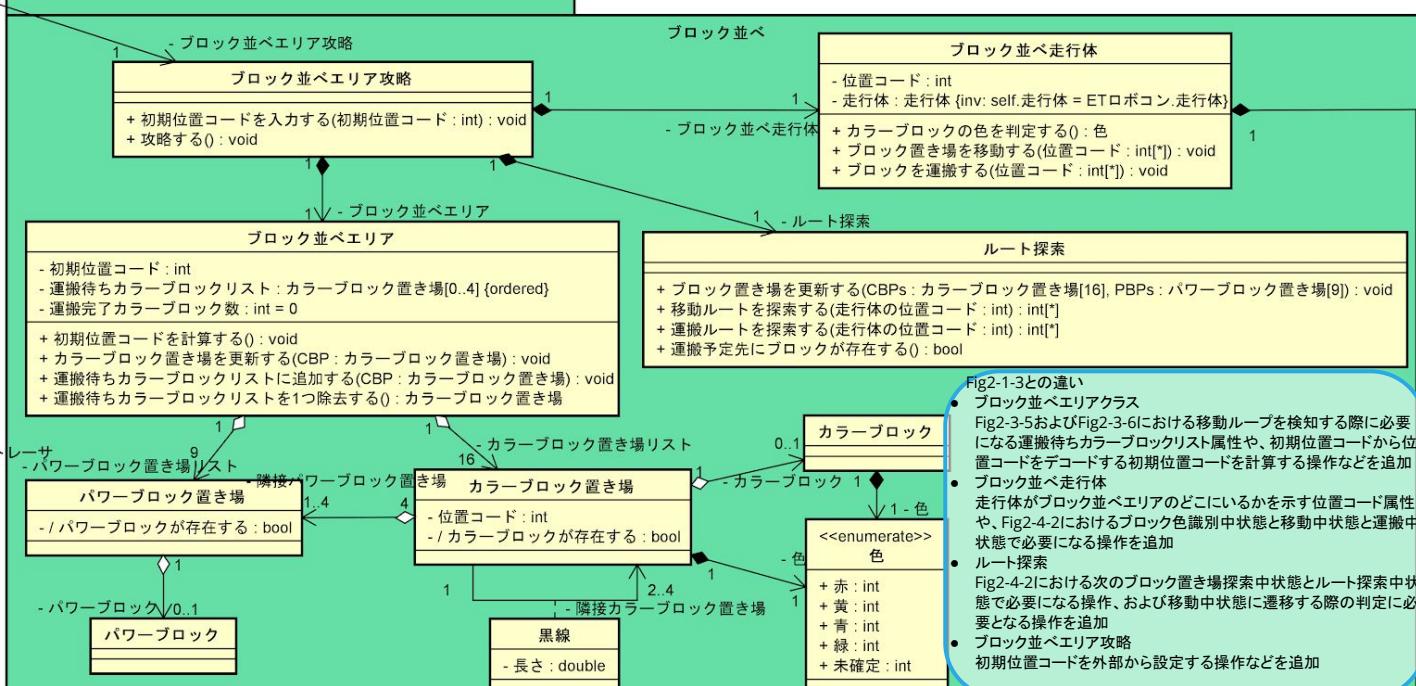


Fig3-2-1 構造のクラス図



4-1. メイン処理

メイン処理の流れをFig4-1-1に、競技全体の走行体の状態遷移をFig4-1-2に、それぞれ示す。各オブジェクトはETロボコンクラスのオブジェクト生成時に生成する。初期位置コードについても、ETロボコンクラスがRコースクラス

を経由してブロック並ベエリア攻略に設定する。これにより、ETロボコンクラスが直接ブロック並ベエリア攻略オブジェクトを保持する必要性を無くすことができ、クラス間の結合度を小さくできる。

Fig4-1-1における操作1.1、操作2.1、操作2.2、操作2.3は、それぞれ、Fig4-1-2における初期位置コード入力中状態、Rコース走行エリア攻略中状態、ブロック並ベエリア攻略中状態、直角駐車攻略状態に対応する。

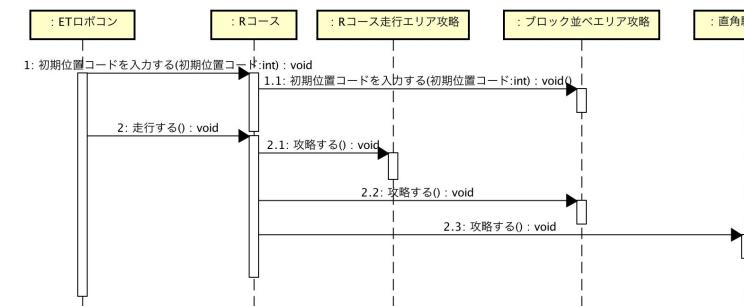


Fig4-1-1 メイン処理の流れ

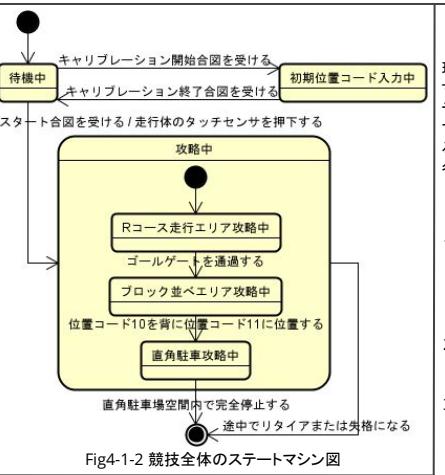
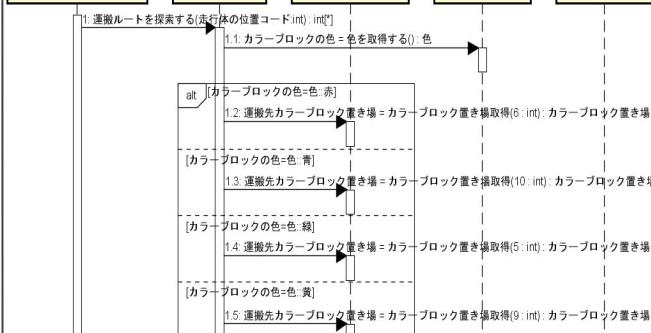
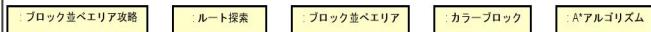


Fig4-1-2 競技全体のステートマシン図

4-3. ブロック並ベエリア攻略詳細処理

ブロック並べを実現するための詳細な処理の流れをFig4-3-1~Fig4-3-2に示す。ここでは、4-2節で説明しきれなかった部分の具体的な処理の流れを分析している。



4-2. ブロック並ベエリア攻略処理

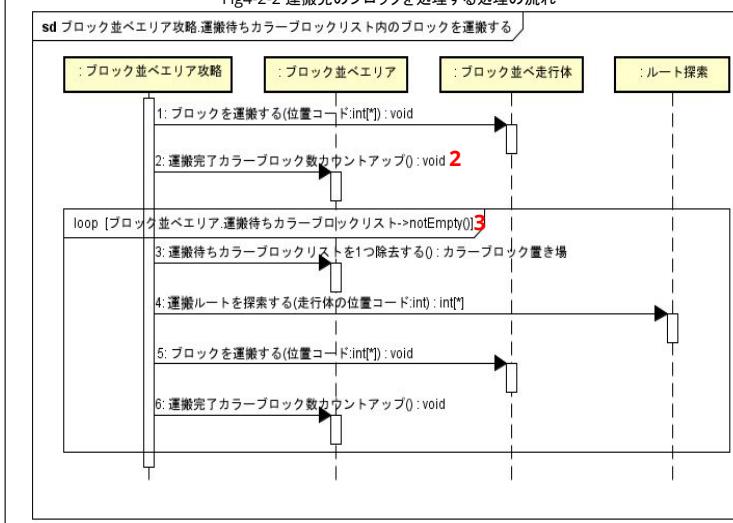
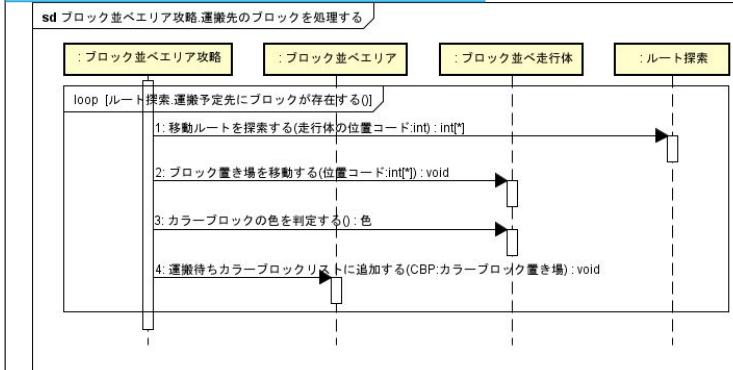


Fig4-2-3 運搬待ちカラーブロックリスト内のブロックを運搬する処理の流れ

Fig4-3-1 運搬ルートを探索する処理の流れ



Fig4-3-2 移動ルートを探索する処理の流れ

5-1. 自己位置推定

2つあるタイヤの回転角から、車体の方位、走行距離、自己座標を周期ごと(4ms)に測定する。Fig5-1-1に車体平面図を示す。

dM_r, dM_l は、微小時間でのタイヤ前進距離

$$d\theta = \frac{(dM_r - dM_l)}{T}$$

$$dD = \frac{dD}{2}$$

$$R = \frac{dD}{d\theta}$$

$$dX = R \cdot \sin(d\theta)$$

$$dY = R \cdot \{1 - \cos(d\theta)\}$$

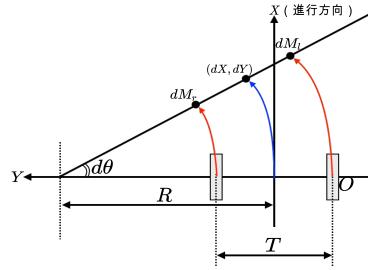


Fig5-1-1 車体平面図

5-2. 初期位置コード入力

キャリブレーション準備の際、初期位置コードが公開される。この初期位置コードをEV3本体液晶に表示される数字を見ながら、上下左右ボタンで入力する。入力完了後、中央にあるボタンを押下し初期位置コードの決定を行う。Fig5-2-1に入力手順を表した図を示す。

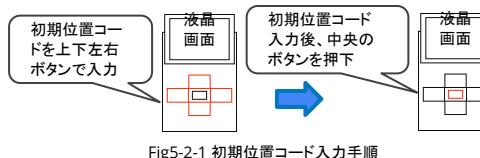


Fig5-2-1 初期位置コード入力手順

5-3. アームブロック保持

アーム制御では、急発進や後退や旋回を行う場合、ブロックの転倒や、アームからブロックが離れてしまう恐れがある。アームからブロックが離れる際の具体的な挙動をFig5-3-1に示す。この問題を防ぐために、アーム使用時には以下の制限を設ける。

減速度120mm/s²

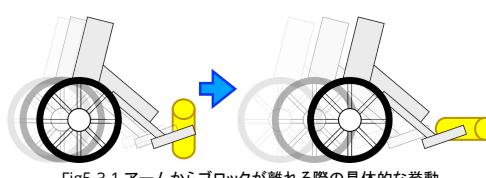


Fig5-3-1 アームからブロックが離れる際の具体的な挙動

5-4. AIアンサーの数字判別

AIアンサーは、各数字の特定箇所を読み取ることで、数字を判別する。左出題数字では、Fig5-4-1に赤丸で示す6つ、右出題数字では、Fig5-4-1に青丸で示す7つを読み取る。黒である箇所は、左出題数字、右出題数字それぞれ、0~7で同じになることはないで判別が可能である。例として、Fig5-4-1に左出題数字が2、右出題数字が3の場合を示す。

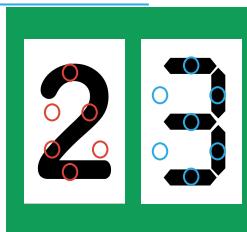


Fig5-4-1 AIアンサーの数字を判別する箇所

5-6 HSV色空間を利用した色識別

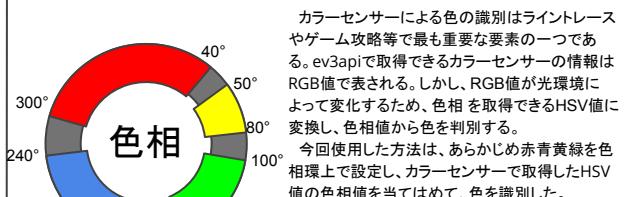


Fig5-6-1 色相環と色の割り当て

5-5. カルマンフィルタを用いた旋回制御

5-4節で述べたAIアンサーの特定箇所を読み取るために、走行体を正確に特定箇所まで移動させる必要がある。従来の走行体の旋回制御では、サーボモーター内のエンコーダーの値から旋回角度を算出している。しかしながら、エンコーダーの値はint型で、かつ外乱の影響を受けるため、正確な旋回制御には使用できない。そこで、式5-5-1に示す状態方程式の角速度 ω をカルマンフィルタ(状態観測器の一種)によって加算し、角度を推定する。このときモーターの入力となるPWM値のシステムノイズ v とエンコーダーの出力となる角度(式5-5-2参照)の観測ノイズ ε も考慮に入れる。ただし、ノイズはガウス分布に従うホワイトノイズとする。

カルマンフィルタを用いた旋回制御のプロック線図をFig5-5-1に示す。カルマンフィルタでは、サーボモーターと同じシステムをシミュレーターとして作成し、状態を更新する。さらに入力として、実際のエンコーダーの値も取ることで、出力の誤差をフィードバックする。カルマンフィルタを用いる場合の状態方程式を式5-5-3に示す。ここで K は、カルマンゲインと呼ばれる定数である。

$$\dot{x} = Ax + Bu + v \\ = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha \end{bmatrix} u + v \quad (5-5-1)$$

$$y = Cx + \varepsilon \\ = [1 \ 0] x + \varepsilon = \theta + \varepsilon \quad (5-5-2)$$

$$\dot{x} = A\hat{x} + Bu + K(y - \hat{y}) \quad (5-5-3)$$

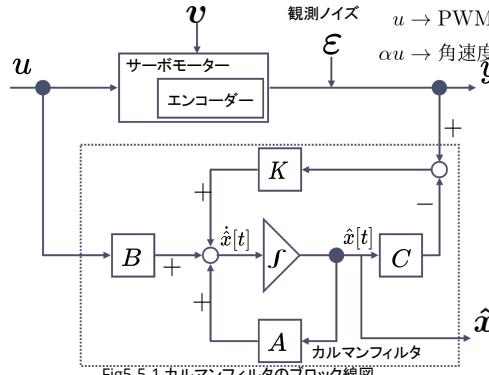


Fig5-5-1 カルマンフィルタのプロック線図

5-9. ログファイル生成

開発を行う際に、EV3RT本体の状態を知りたい場合がある。そこで、バッテリーの電圧や電流などの状態(Fig5-9-1参照)をSDカードの「Log」フォルダにCSV形式のログファイルに出力する。また、ログファイルが上書きされないように、出力ファイル名を前回保存したファイル名と被らないようにした(Fig5-9-2参照)。

Voltage	Ampere	leftMotorCounts	rightMotorCounts
8357	48	352	353
8337	54	354	356

Fig5-9-1 ログファイルより一部抜粋

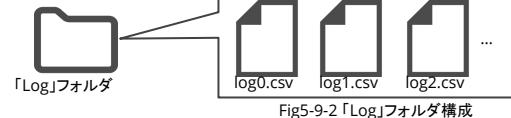


Fig5-9-2 「Log」フォルダ構成

5-7. 色識別の誤検知防止

色識別にはいくつかの課題が存在する。

課題1. 黒と白は色相を持たない(彩度と明度によって決まる)ため、色相のみで色を識別すると、白と黒の上で誤検知を起こしてしまう。そこで、明度が極端に高い場合に白と判断し、極端に低い場合は黒と判断することで、色識別の誤検知の可能性を減らすことができる。黑白判定の仕組みをFig5-7-1に示す。

課題2. 走行中の走行体はセンサーと地面の距離を一定に保つことは難しい。そのため、誤検知してしまう可能性がある。対策として、循環パッファを用意し、色検知を行うたびにパッファに色を格納していく、多数決を取り、最大数の色を採用する。循環パッファを利用した誤作動防止方法をFig5-7-2に示す。

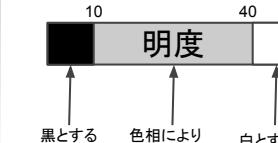


Fig5-7-1 黒白判定の仕組み

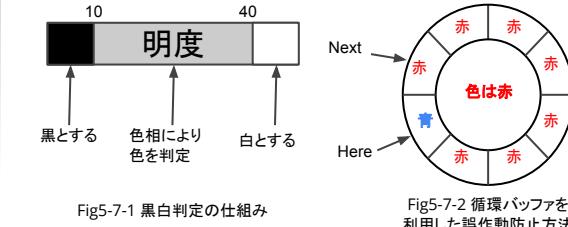


Fig5-7-2 循環パッファを利用した誤作動防止方法

5-8. ノイズ除去フィルタ

カラーセンサーを使用する際に、センサー値をそのまま使用すると外乱(ノイズ)の影響を受けて誤動作を起こす可能性がある。そこで、誤動作を最小限に抑えるためにローパスフィルタを実装し、PID制御で用いた。ローパスフィルタを用いることで特定の高周波数をカットでき、センサーの外れ値に強くなる。今回使用するローパスフィルタの式を式5-8-1に示す。

ローパスフィルタ以外にも様々なフィルタが存在する。今回ローパスフィルタを用いたPID制御のプロック線図を5-11節に示す。

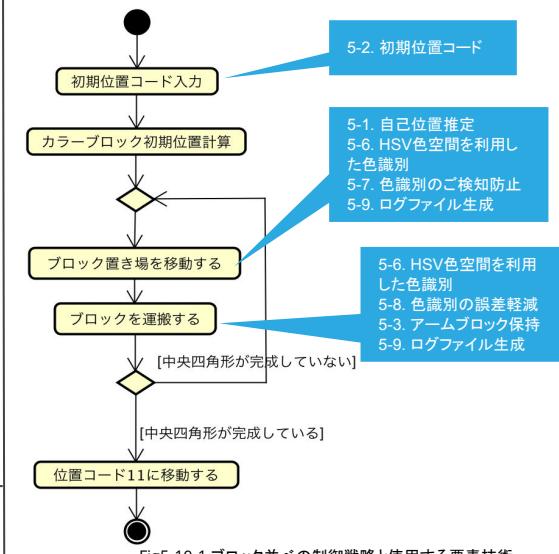
$$y_i = 0.9 \times y_{i-1} + 0.1 \times x_i \quad (5-8-1)$$

y_i → 出力値

x_i → センサー値

5-10. ブロック並べの制御戦略

ブロック並べの制御戦略と使用する要素技術をFig5-10-1に示す。



5-11. ライントレース時の走行制御

5-8節で示したノイズ除去フィルタを用いて、ライントレース時の走行制御においてPID制御を行った。Fig5-11-1にPID制御のプロック線図を示す。

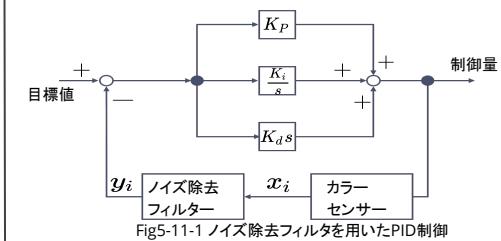


Fig5-11-1 ノイズ除去フィルタを用いたPID制御