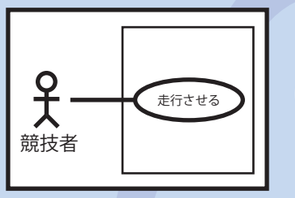
1 ユースケース図



他のユースケースは？

キャリブレーションは
ユースケースじゃないの？

ユースケースは「走行させる」1つだけ！

走行させるユースケースでは大きく分けて、

1　線のどちら側を走るかや閾値の読み込みなどの
　　初期設定を行う
2　コースを一周するまで走行する

という２つの作業を行っている。

アクター競技者にとっての目的は

「パスファインダがコースを走る」

ということ。

そして初期設定は、走行前に必ずすることだ。

走行しないで初期設定だけするということはありえない。

だから、キャリブレーションは走らせる
ユースケースの中に含まれるので、

ユースケースは

「走行させる」

一つだけだと考えた。

変更が容易な設計で
最高の走りを実現！

2 走行戦略

直線では首をなるべく振らないようにすれば速く走れる。

カーブではコースアウトしないよう、しっかり首を振りたい！

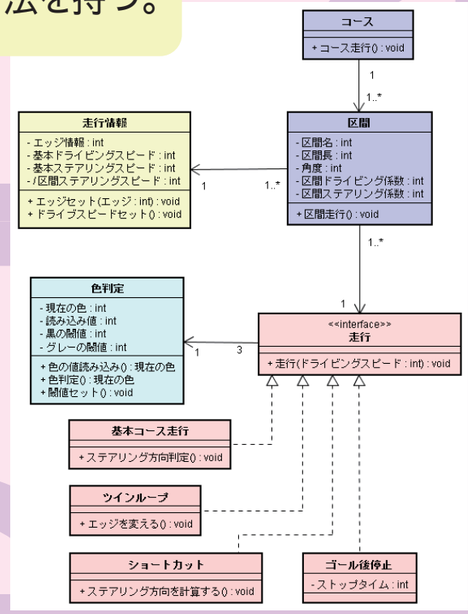
難所では特別な走行方法に切り替えなくては…

コースの場所によって違う走り方が必要

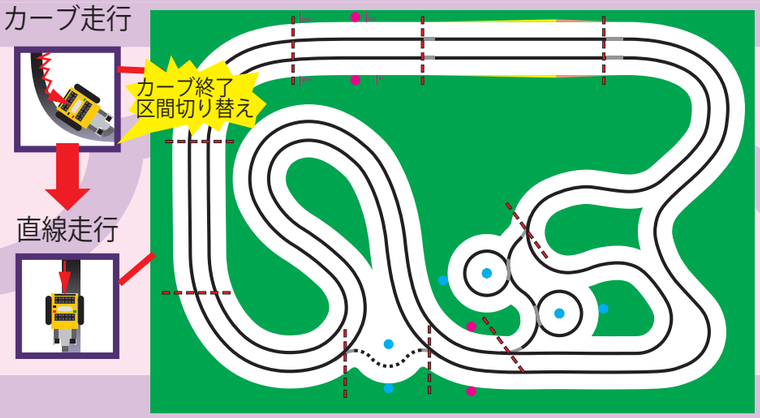
コース情報を持てば良い！

コースは区間から成り、各区間は最適な走行方法を持つ。

タイマーやグレー検出により区間を切り替え、
コースの形状にあった走り方をすることが可能！！



カーブ走行



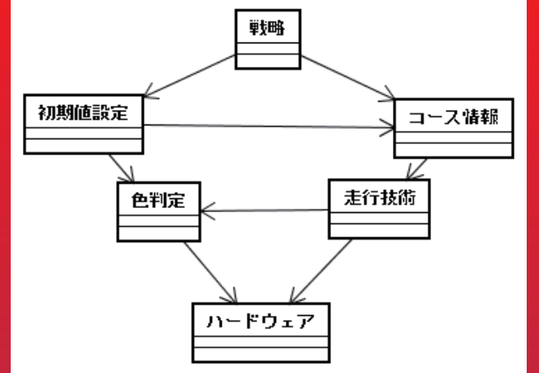
直線走行

コースが持つ区間を変更すれば色々なコースで走れる

ツインループの動作が少し不安…	→	区間を変更して、ツインループの無いコースに
新しい走行アルゴリズムを考えた！	→	区間が持つ走行を変更するだけで OK
コースの形は違うけれど走り方は一緒	→	カーブ用走行のインスタンスは一つだけ。 各区間ごとにカーブの長さやステアリングスピードを変更すれば良い

3 クラス図

概念クラス図



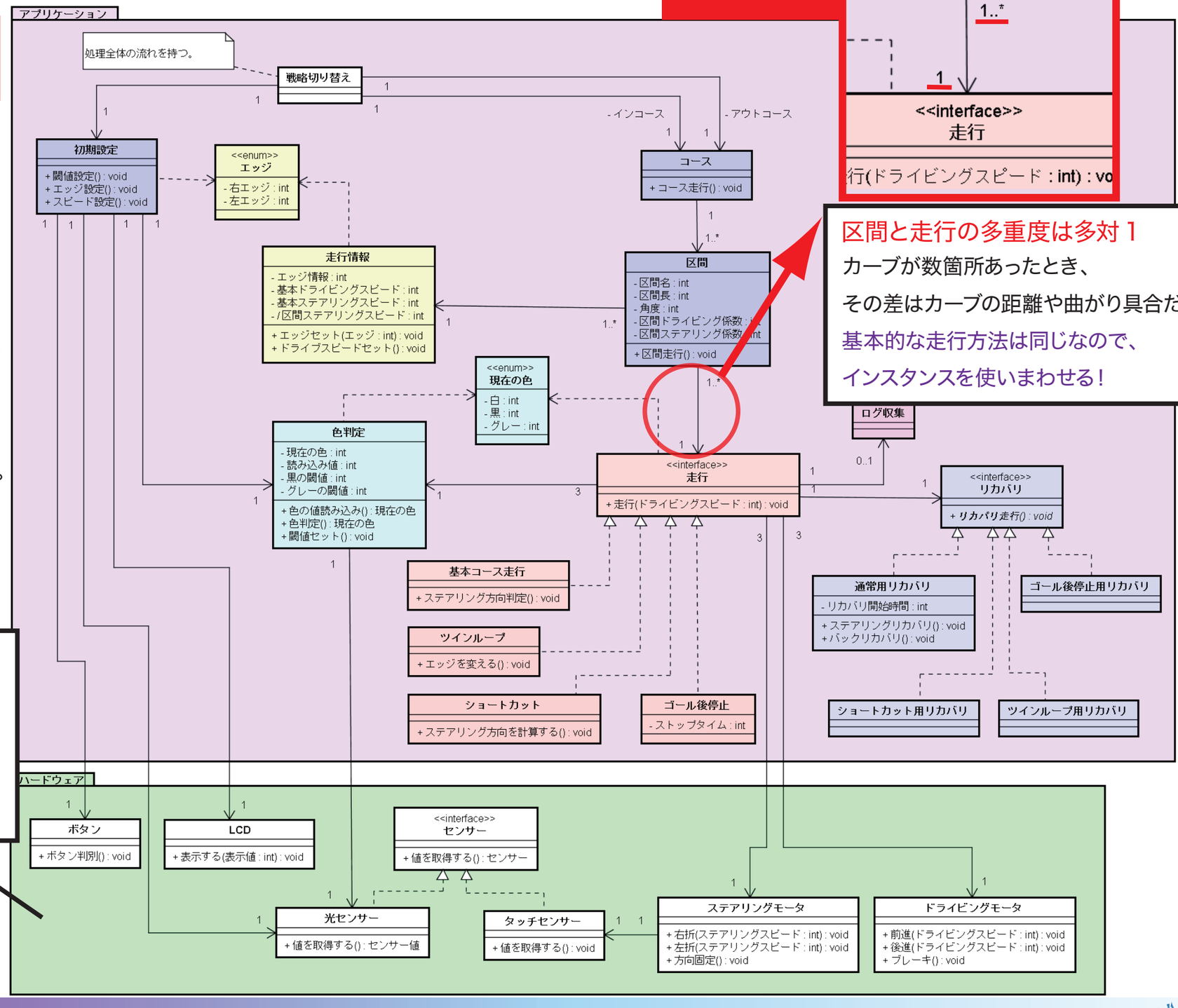
戦略クラスは、初期設定とコースを開始させる全体を統括するところ。
設計時には Factory クラスの生成も担当した。
初期値設定・コース情報・走行技術・色判定がアプリケーションパッケージの主な要素だ。

操作を分かりやすくするための
ラッパークラス

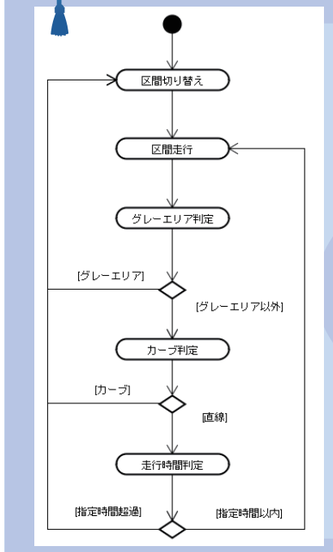
ハードウェアにアクセスするための
クラスをまとめた。
右折･左折など分かりやすい名前を
付け、呼び出しやすくしている。

ポイント

区間と走行の多重度は多対 1
カーブが数箇所あったとき、
その差はカーブの距離や曲がり具合だけ。
基本的な走行方法は同じなので、
インスタンスを使いまわせる！



4 実装方法



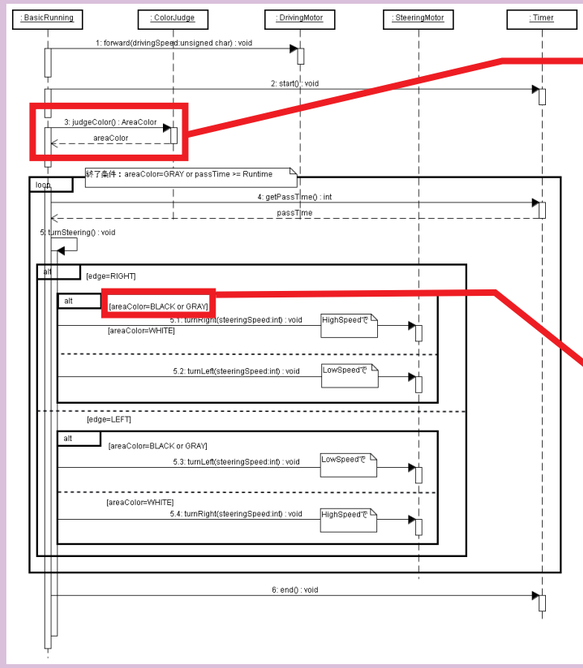
区間の切り替え方

区間は直線、カーブ、難所に分けられている。

これらの区間をコースクラスが認識し、区間を入れ替えていく

どうやって
入れ替えているの？

- 直線・カーブの切り替えはカーブの角度を検出
- 難所は手前にあるグレーを検出する事で切り替え
- さらに、各切り替えポイントの検出が失敗したとき用に
  区間走行の経過時間による切り替えで保障



直線走行

judgeColor メソッドからは、

「現在自分が何色のエリアにいるか」

が返ってくる。

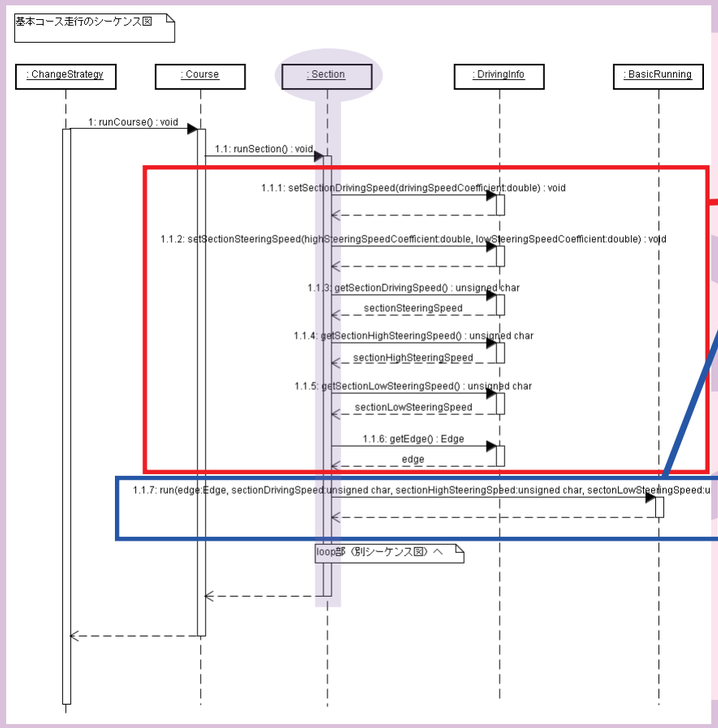
この色情報によって、首振りの

方向を決める。

区間分けによってグレーが検出される場所は切り離されているので、

ここでは、グレー検出時の特別な処理は必要無い。

→直線を走ることだけに
専念できる



区間・走行・走行情報の関連

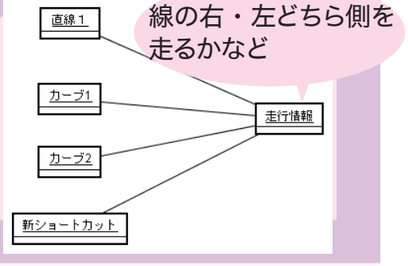
セクションの役割は、

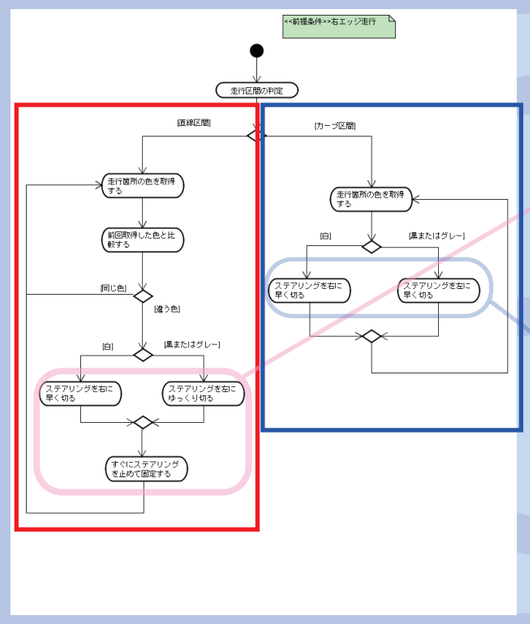
1 走行情報に設定をセットする

2 走行クラスに情報を渡して
走行を開始させる

の2つである。

走行情報には全コース共通の
情報が入っている





直線走行と カーブ走行の違い

直線区間では、ステアリングの

首振りを少なくしてなるべくまっすぐに

走るようにする。

前回の色情報を記憶し、

色が変わったときだけ首振りをする

カーブ区間では、コースアウトしないように、細かく首を振る。

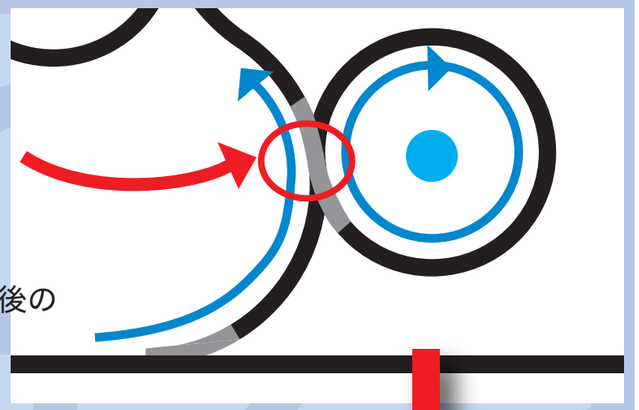
直線のような記憶方式をせず、

色を判定するごとに

高速でステアリングを切る

5 難所走行

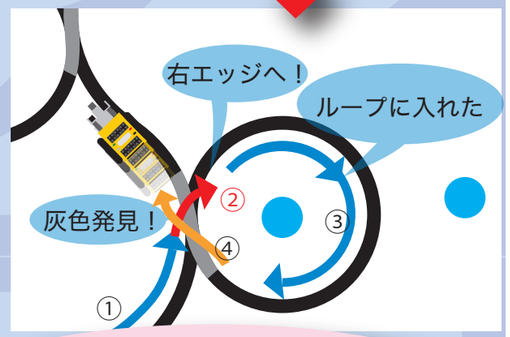
ツインループ



ツインループ区間は
グレーがポイント！

＊何もしないとループに入れない

＊ループに入った後もグレー検出後の
アクションが無いとぐるぐる
回ってしまう･･･



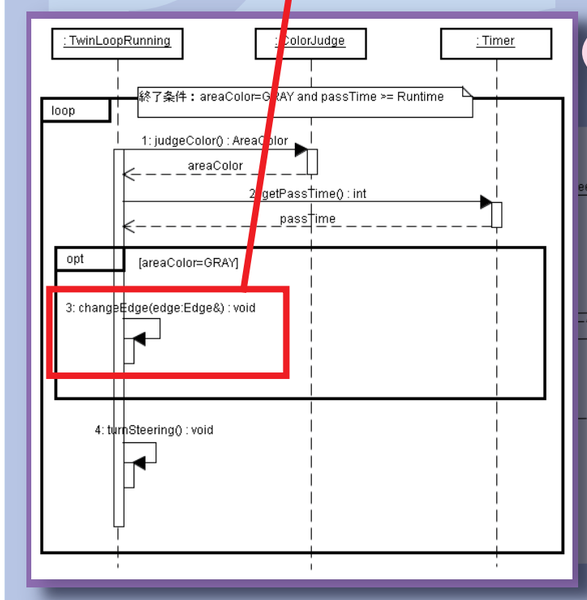
①最初はライン上、左エッジを走行

②最初のグレーを検出したら、右エッジに
変える　→ループ進入

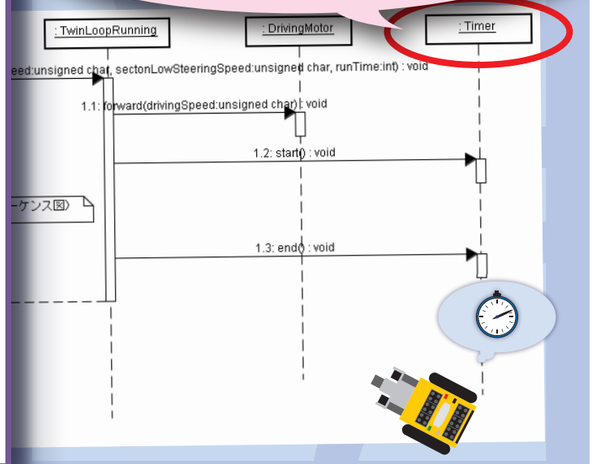
③ループを一周する

④グレーがきたら再度左エッジ走行に

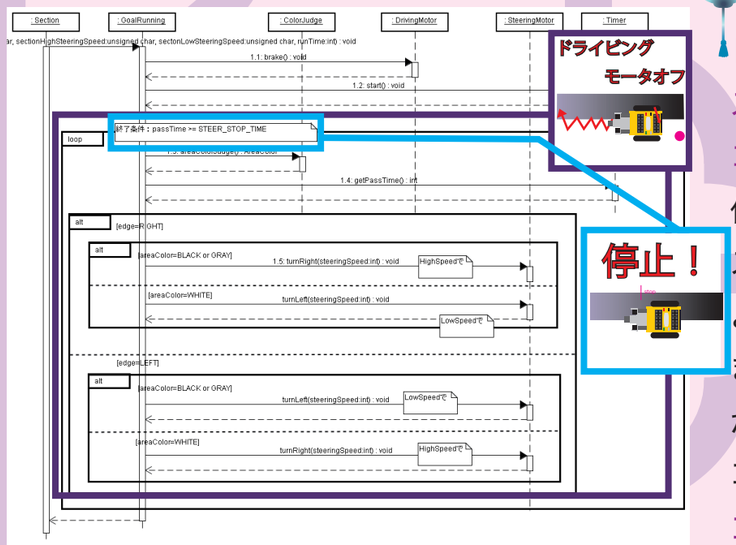
走行する線の左右エッジを変えることで攻略！



グレー検出＆タイマーによる
経過時間チェックで
ツインループ区間の終了を判定



ゴール後停止

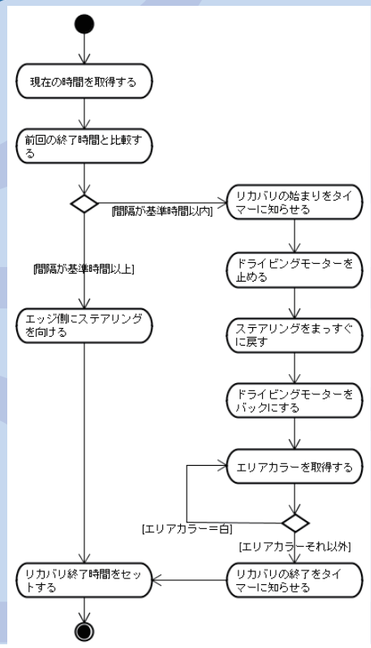


ドライビングモータを止めた後も、
ステアリングだけは振り続けて
コースアウトしないようにする
仕組み

ステアリングの停止はタイマーに
よる制御で行う

また、ステアリングの振れ幅
からカーブを検出すると、
ゴールラインを通り過ぎたと判定し
ゴールまでバックするリカバリ機能
もある

6 リカバリ



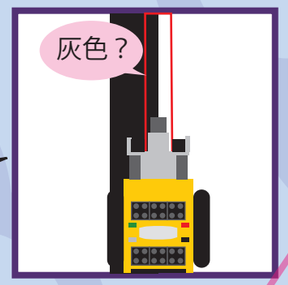
より効率の良いリカバリのために、
２段階リカバリ機能を搭載！

１段階目：前回のリカバリからの経過時間が長い
→ コースアウトした直後の特徴
光センサーが誤判定を起こした直後
ステアリングを逆にするだけ　高速

２段階目：前回のリカバリからの経過時間が短い
→ 連続してコースアウトが起きている
大きくコースアウトしている可能性あり
一度バックしてコースを探し直す　確実

7 色判定

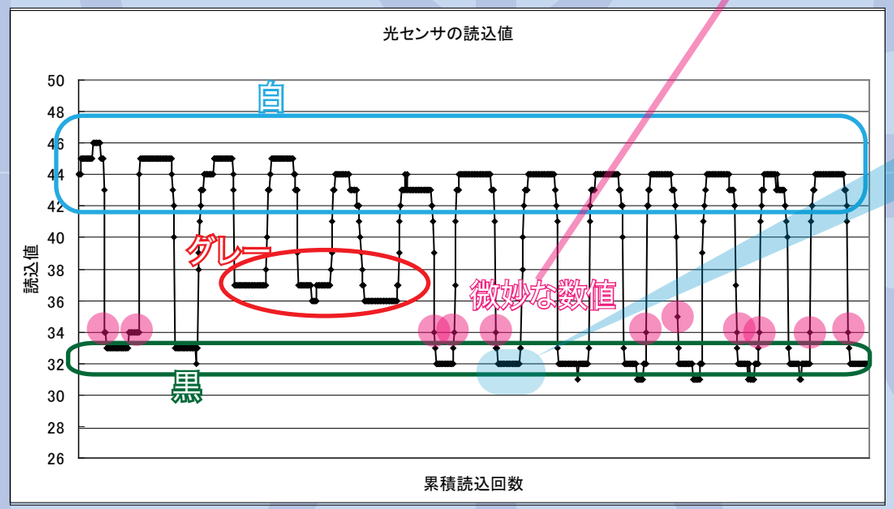
白と黒の中間 ＝マーカー部分



白と黒の境界領域とマーカの違いを区別する方法が問題。

どうちがうんだろう…？

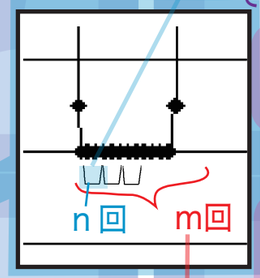
そこで、試走会でとった光センサー値の走行ログを解析した。



グラフを見ると特徴が分かってきた。

確実に色判定をするには、
安易に光センサの読込値を信じてはダメ。

そこで**3 段階判定方式**を採用した！
白・黒だけでなく、
グレーの確実な検出・白と黒の境界領域における誤判定の回避を実現！！



1 中途半端なセンサ値は信用しない！

光センサが各色を読み取っているとき、値は安定している。
各色の間にある微妙な数値は前回出た色と判断する。

誤判定が減って精度アップ！

2 直前n回の判定情報を記憶しておいて、多数決をとる

連続して同色の値を読んだときは確実にその色。
一個のセンサ値のみで判定するとグレーと誤認識される

でも…

十分な判定精度を得るためには
nをある程度大きくする必要がある。

nが大きすぎると、ステアリングを
切るタイミングが遅くなり、走行が不安定に。

nの値はどうしよう？

ステアリングを切る判断が
遅れるわけにはいかない！

3 グレーの判定基準を２段階にする！

m回連続してグレーが検出されたとき
にのみ、マーカ上にいるとみなす。

これで誤判定が無くなった！