1. 要素抽出

概要：「ベーシックステージを走行する」機能を実現します

1.1 「ベーシックステージを走行する」機能の概要

ベーシックステージ攻略のコンセプトは、

「確実に完走する」です。

確実に走行するためにライントレース時、エリアでカーブとストレートが判断し適切な速度とカーブ補正処理を行います。また、ラインからの脱線を想定し自動的に脱線を検知して復帰を行います。

そして、安定した制御の中で走行タイムを縮めるために、ショートカットを行います。

ショートカットの有用性を検証する

ショートカットの実験：エリア5を直線で走ると2秒程度だった。ライン復帰のオーバーヘッドを考えて3秒と想定

ライントレースの実験：通常のライントレースを行い、ショートカット開始地点から、終了地点までのタイムを計測

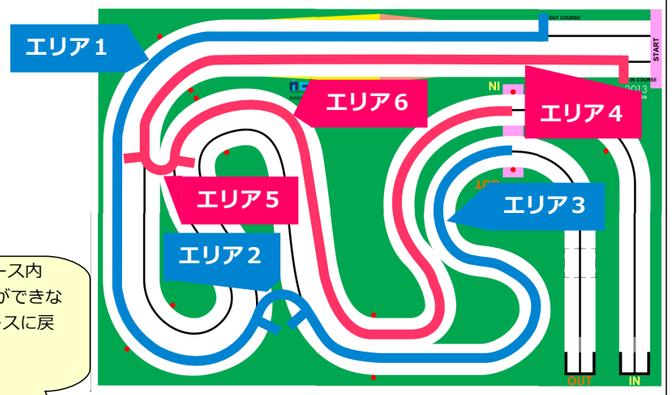
実験の結果

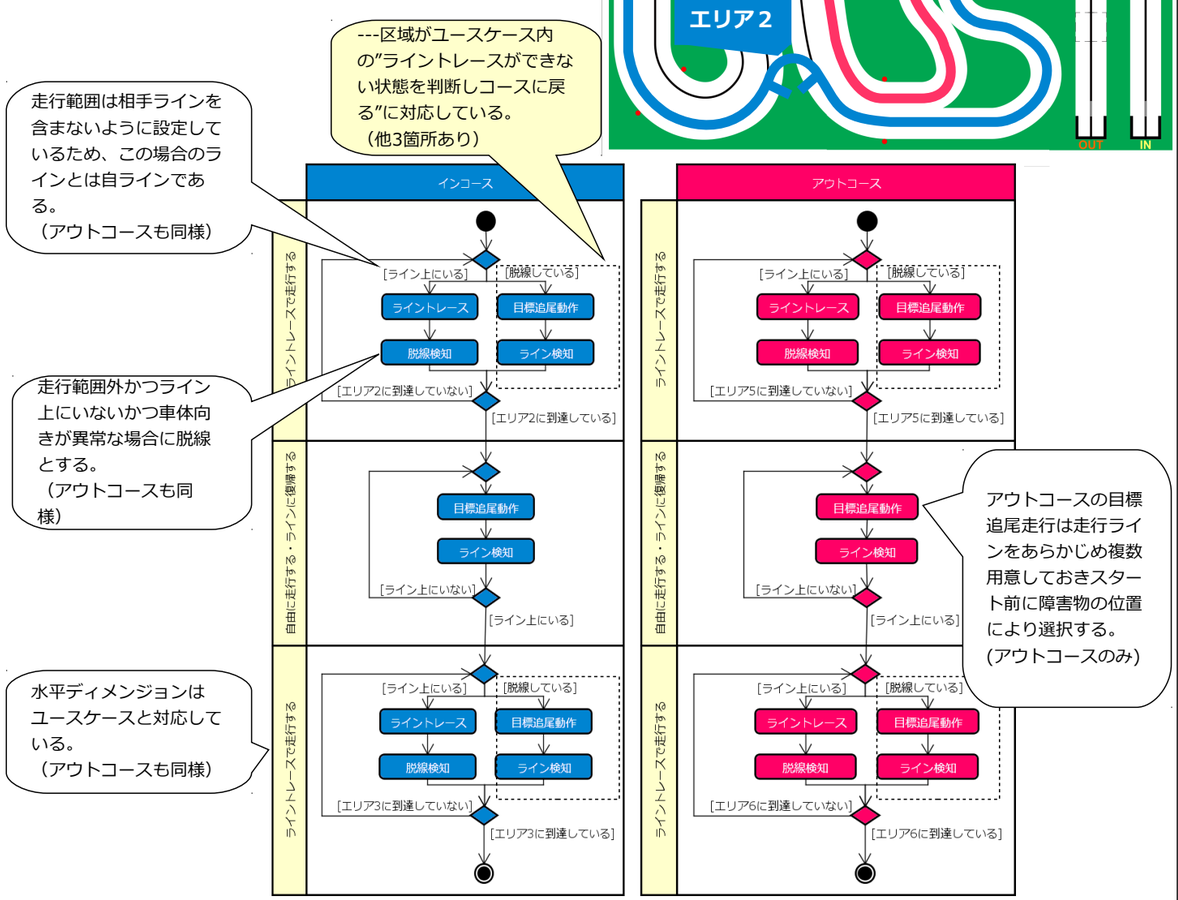
ショートカットの場合	=	約3秒
ライントレースの場合	=	約15秒
中間ゲートのボーナス	=	5秒

3秒ー（15秒ー５秒）＝ー7秒の短縮になる

1.2 実現手順の検討

1.1で記述している機能の実現手順をアクティビティ図を使用してを検討する。
インコースの走行戦略は水色、アウトコースの走行戦略はピンク色で記載している。また、アクティビティ図の水平ディメンジョンは後述するユースケースと対応している。





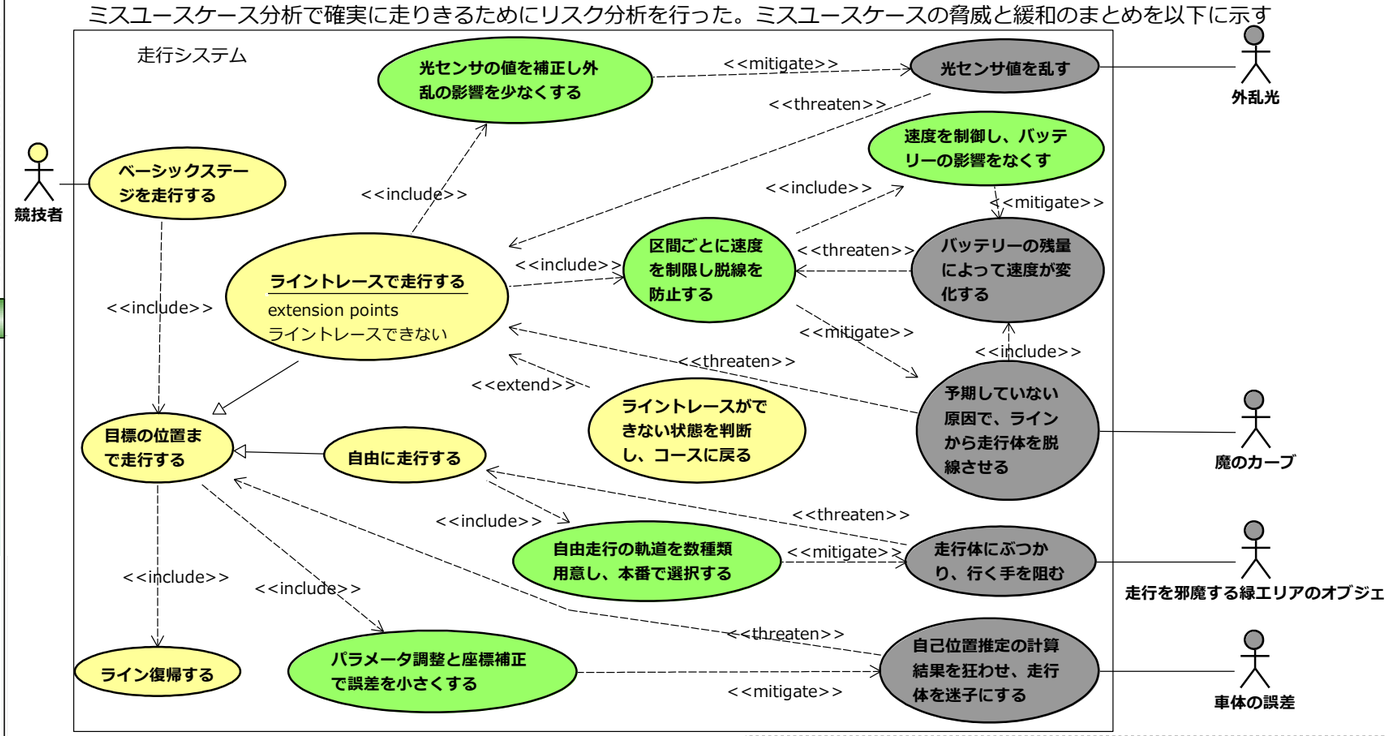
1.3 開発の効率化

開発段階での効率を上げるためホストマシン、シーケンス走行、走行データのロガーを実装する。

ホストマシン(外部PC) ⇒シーケンスの実行、データの取得を容易にする。
シーケンス走行 ⇒ハードコーディングなしで動作の作成を可能にする。
走行データのロガー ⇒取得したデータを下に動作の分析やパラメータの調整を可能にする。

1.4 ミスユースケース分析によるリスクの洗い出しと緩和策の検討による要素の抽出

ミスユースケース分析で確実に走りきるためにリスク分析を行った。ミスユースケースの脅威と緩和のまとめを以下に示す



ユースケースは1.2のアクティビティ図の水平ディメンジョンの項目と対応しているため詳細なユースケース記述は省略する。ミスユースケースは簡易記述を記載する。

凡例：ユースケース／緩和ユースケース／ミスユースケース

名称	光センサ値を乱す	バッテリーの残量によって速度が変化する	予期していない原因で、ラインから走行体を脱線させる	走行体にぶつかり、行く手を阻む	自己位置推定の計算結果を狂わせ、走行体を迷子にする
脅威	蛍光灯のノイズや、会場の明るさによって、光センサの値を乱し、安定した走行ができなくなる。	区間ごとに走行体の速さを設定しても、バッテリ残量によって同じ速度にならない。	2013年度の魔のコーナのように、対策を行っていたも、走行体をラインから脱線させる。	緑エリアのショートカット時に、走行体にぶつかり、予定通りの自由走行ができない。	車体の誤差や、コースのゆがみによって、自己位置推定の結果を狂わせ、自己位置の結果を乱す。
緩和	ローパスフィルタ処理でノイズを減らし、会場の明るさを光センサ値の非線形補正、正規化などを施し外乱による影響を小さくする。	目標速度に制御することで、バッテリーによる速度の影響を無くす。	自己位置推定の結果からカーブのエリアを検知し、速度を落として走行する。	ショートカット時の走行ルートを数種類用意しておき、本番でエリアのオブジェにぶつからない走行ルートの選択を行う	走行のログデータからタイヤ径と車幅の定数を調整し走行時は座標の補正を行う。

＊抽出したベーシックコース攻略要素の一覧

上記ユースケース図のユースケース記述をもとに、実装する機能を抽出した。抽出した機能のうち、走り方を選択する処理とパラメータの算出処理は、ベーシックステージ走行中には動作しないため、概要の記載のみとする。

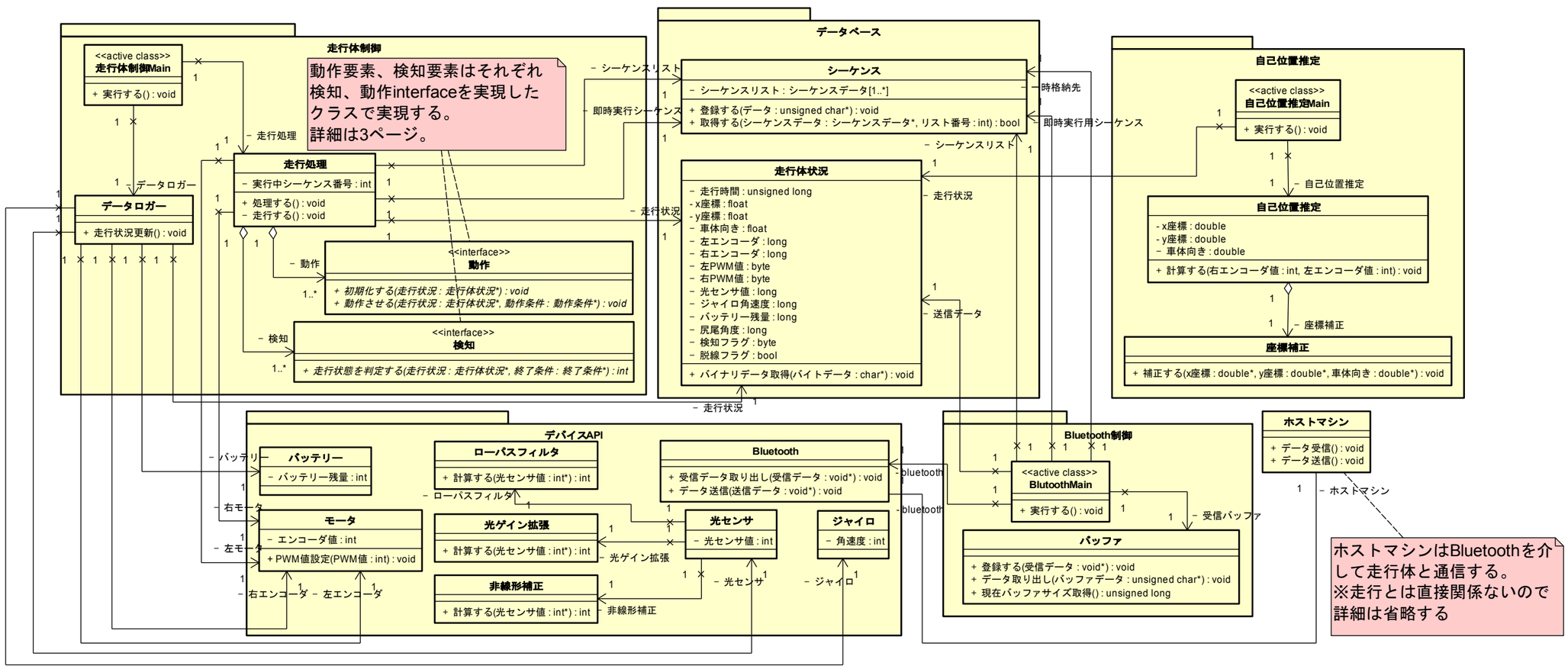
		動作要素		検知要素				処理要素								
		ライントレース動作	目標追尾動作	エリア検知	ライン検知	走行範囲外の検知	車体向き検知	ローパスフィルタ処理	光センサの非線形補正	光センサ値の正規化	区間ごとの速度制限	速度制御処理	自己位置推定	座標の補正	ログの取得	走り方の選択	自己位置パラメータの算出
ユースケース	目標の位置まで走行する::ライントレースで走行する	○		○									○				
	目標の位置まで走行する::自由に走行する		○	○			○						○				
	ライン復帰する		○	○	○								○				
緩和ユースケース	ライントレースができない状態を判断し、コースに戻る				○	○	○						○				
	光センサの値を補正し外乱の影響を少なくする							○	○	○							
	区間ごとに速度を制限し脱線を防止する										○		○				
	速度を制御し、バッテリーの影響をなくす											○	○				
	パラメータ調整と座標補正で誤差を小さくする												○	○	○		○
	自由走行の軌道を数種類用意し、本番で選択する															○	

2. 全体設計

概要： 全体構造および共通要素について記載する。
※走行処理の詳細は3ページ目に記載する。

2.1 全体構造

プログラム全体の構造を示す。
走行体制御パッケージの動作および検知インターフェースを継承したクラスについては別途記載する。



※クラスの全プロパティにはget/setメソッドを作成する

データ構造

<<struct>> シーケンスデータ
+ 動作種別 : int
+ 検知種別 : int
+ 動作条件 : 動作条件
+ 終了条件 : 終了条件
- 即時実行フラグ : bool
- 走行スタートフラグ : bool
- 初期化フラグ : bool
- 追尾用目標座標転送フラグ : bool

<<struct>> 動作条件
+ 左PWM値 : int
+ 右PWM値 : int
+ ジャイロオフセット : int
+ 光センサ中央値 : int
+ ライントレース用前進速度 : int
+ ライントレース用旋回速度 : int
+ PID制御用P値 : float
+ PID制御用I値 : float
+ PID制御用D値 : float
- 目標x座標 : float[1..*]
- 目標y座標 : float[1..*]
- 目標座標のインデックス更新周期 : long

<<struct>> 終了条件
+ 時間 : long
+ x座標 : float
+ y座標 : float
+ 車体向き : float
+ 左エンコーダ値 : long
+ 右エンコーダ値 : long
- 不等号を決めるフラグ : bool[6]

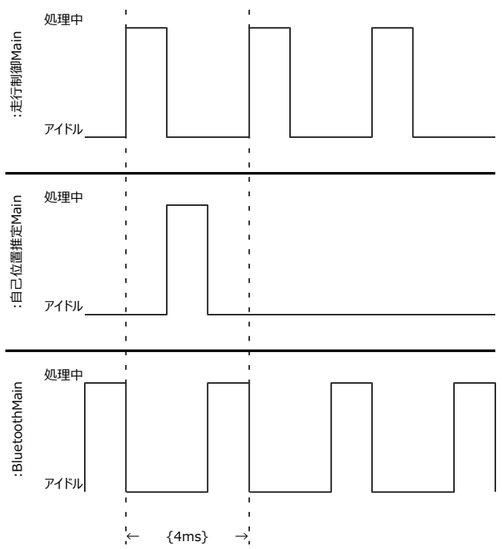
2.2 タスクの実行時間

下記のアクティブクラスはそれぞれ別のスレッド(タスク)で動作する。

- 走行体制御Main
- 自己位置推定Main(40ms周期)
- BluetoothMain

走行体は4msで動作を更新していくため、上記アクティブクラスの合計時間が4msを超えないように実装を行う。

データベースへのアクセスタイミングの重複を避けるため、**各タスク同士を排他処理**にする(スレッドセーフにする)



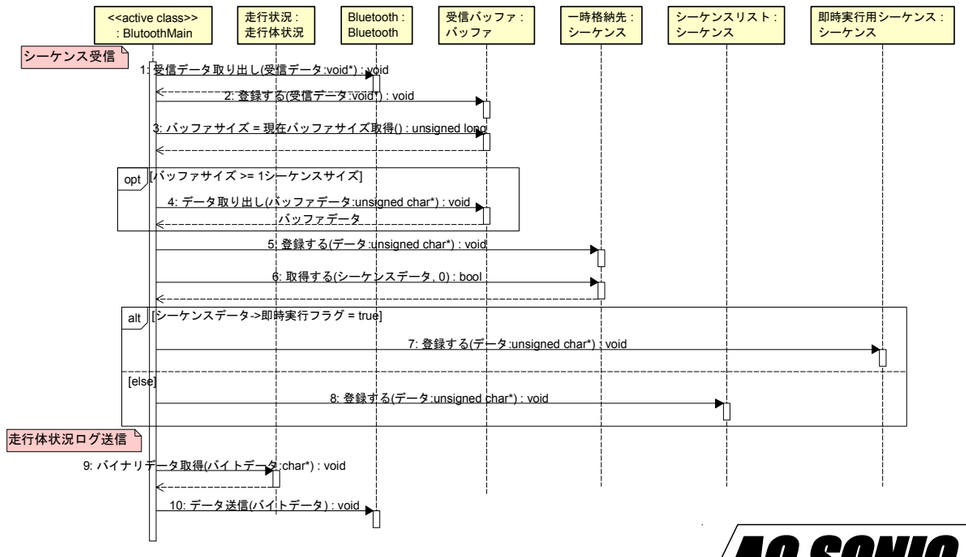
2.3 ホストシステム-走行体間のデータ送受

走行体がシーケンスを受信、ログデータを送信するためにBluetoothを使用する。

[受信処理]
受信するシーケンスには下記の2種類があり、別々のインスタンスに格納する

- 即時実行するためのシーケンス
- 逐次実行するためにリスト化したシーケンス

[送信処理]
ログデータを送信するには、走行状況のプロパティをバイトデータのリスト(バイナリデータ)として受け取り、Bluetoothを介して送信する。

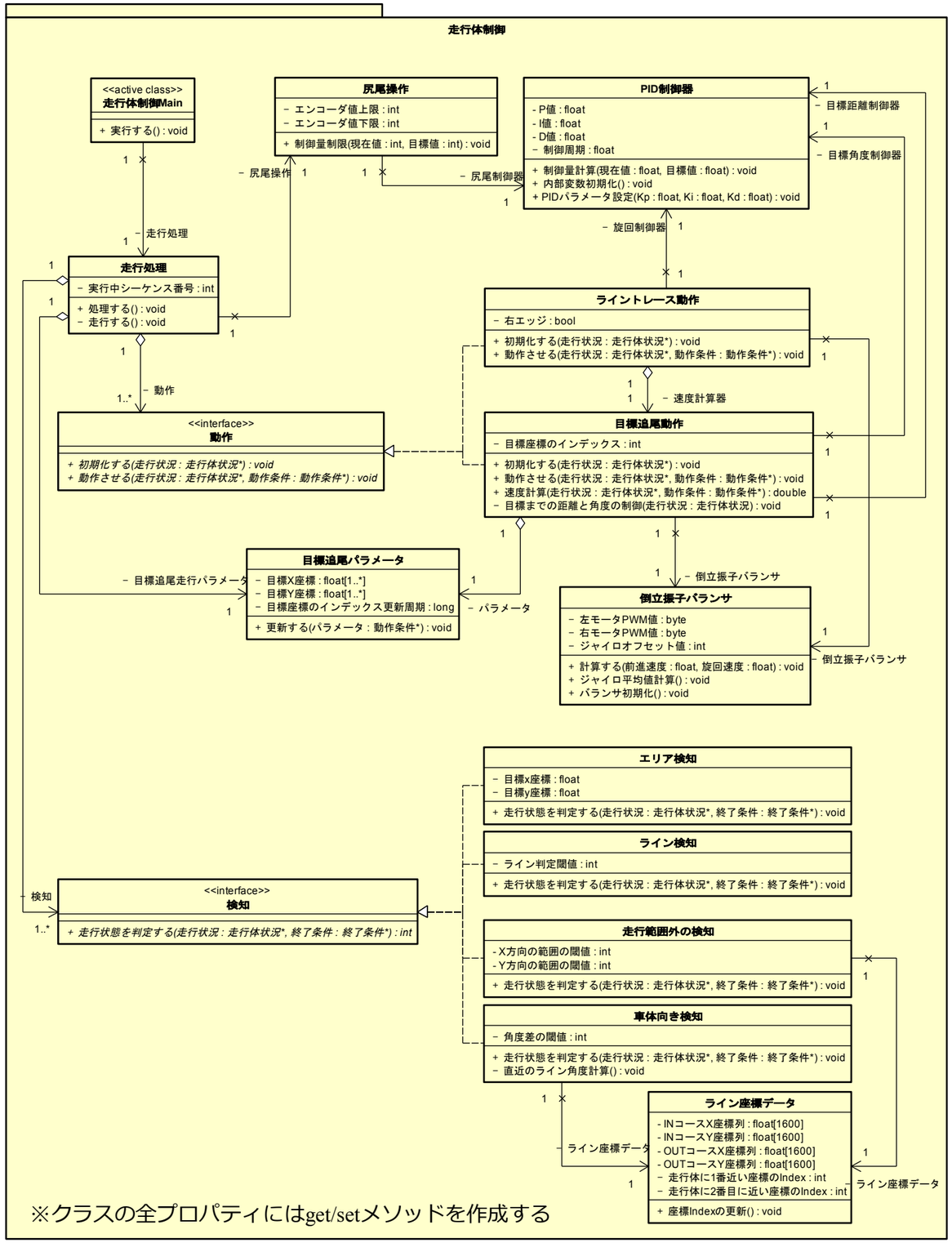


3. 詳細設計

概要： 各ユースケースを実現する走行制御について、各制御個別の構造を記載する。
全体、共通的な構造については2ページを参照のこと。

3.1 走行処理の構造

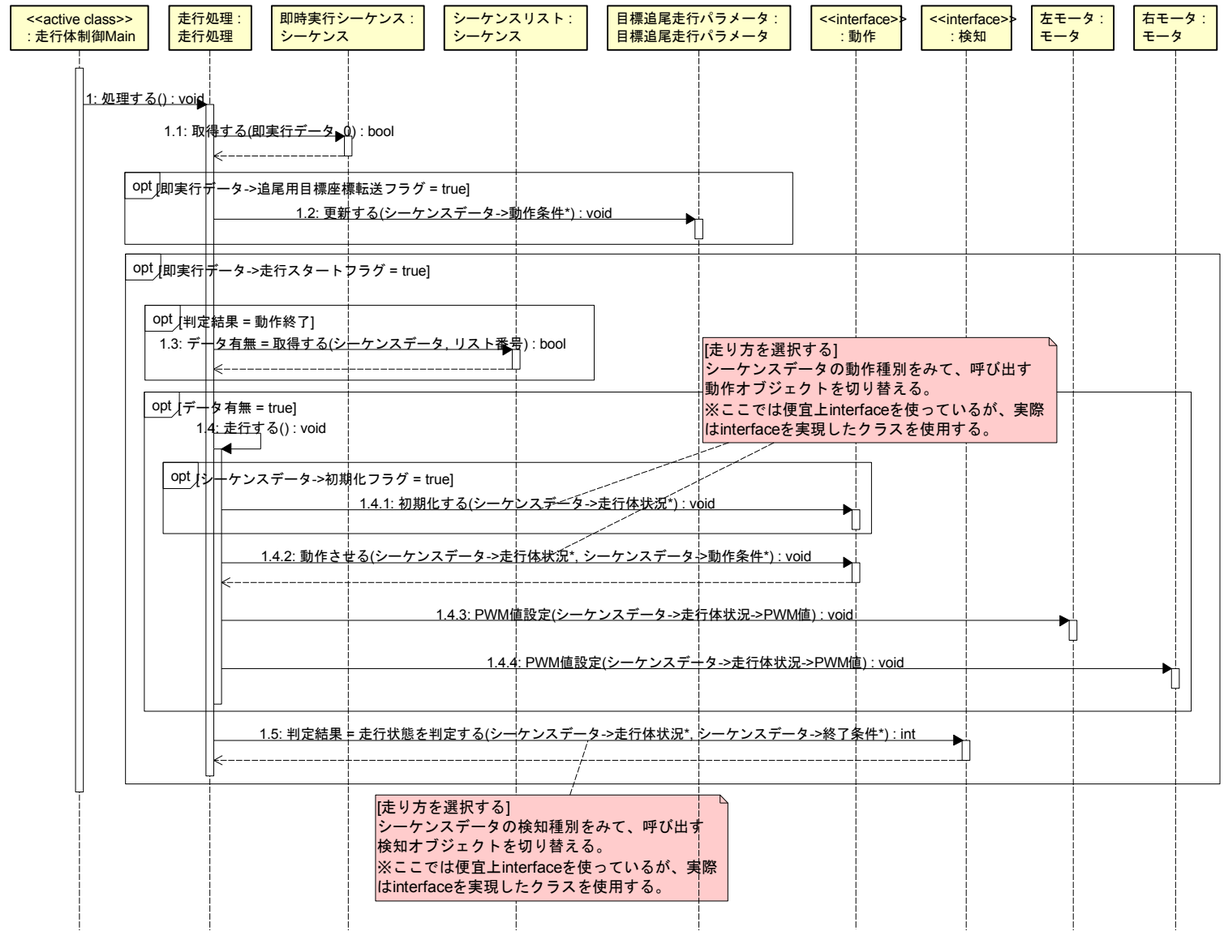
走行処理は動作と検知を組み合わせて行う。
動作と検知はボーナスステージで追加される可能性を考慮し、インターフェースにしている。
下記にクラス図を示す。



※クラスの全プロパティにはget/setメソッドを作成する

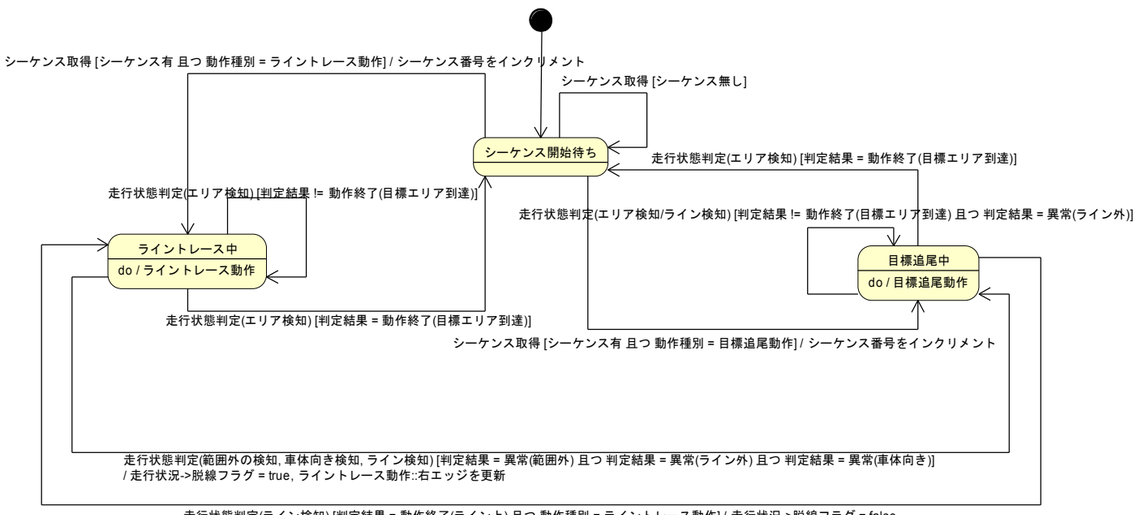
3.2 走行処理の基本シーケンス

走行処理クラスの基本処理を示す。



3.3 走行処理の状態遷移

走行処理クラスの振る舞いを示す。ライントレース時は脱線を検知してライン復帰を行う

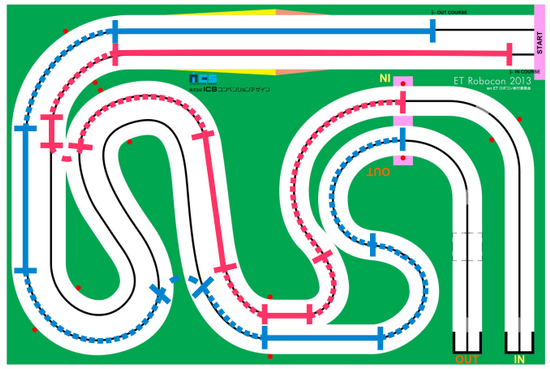


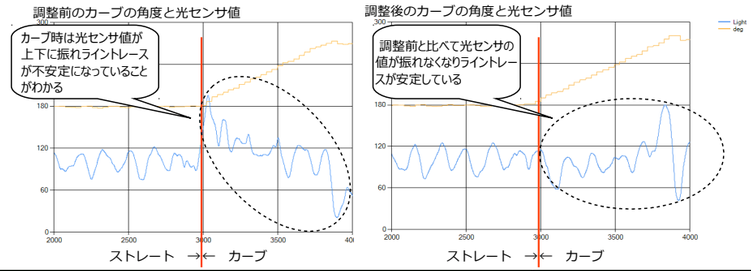
4. 制御戦略

概要： 走行に必要な制御戦略を主に要素技術をアクティビティとするアクティビティ図を使用して記述する。

4.1 区間ごとの速度調整とカーブオフセット処理

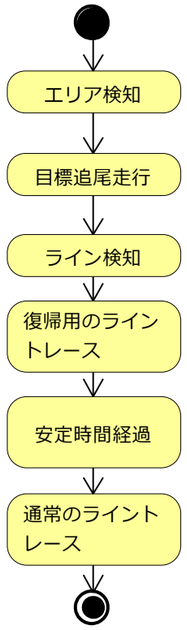
安全にかつ高速にライントレースを行うために区間ごとのカーブオフセット処理と最適な速度調整を行う。カーブオフセット処理は下図ログデータのようにカーブ時のライトセンサの振れ幅が小さくなるように Turn（倒立伸子時の旋回値）にオフセットを加えることで調整する。

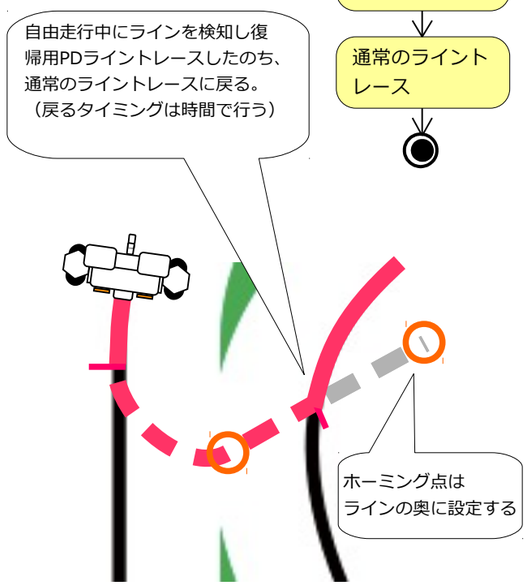




4.2 インコースのショートカット戦略

インコースは障害物が置かれている可能性のある緑エリアを避け、コースの白エリアが接触している箇所でショートカットを行う。ショートカットの開始条件はエリア検知により判定し、目標追尾動作により自由に走行する。到達目標点は自由走行中に1箇所とラインを超えたところに1箇所設置し、ラインを検知したタイミングで復帰シーケンスに移行する。復帰には、専用にPIDを調節したライントレースを使用し、安定時間経過後、通常のライントレースに戻る。

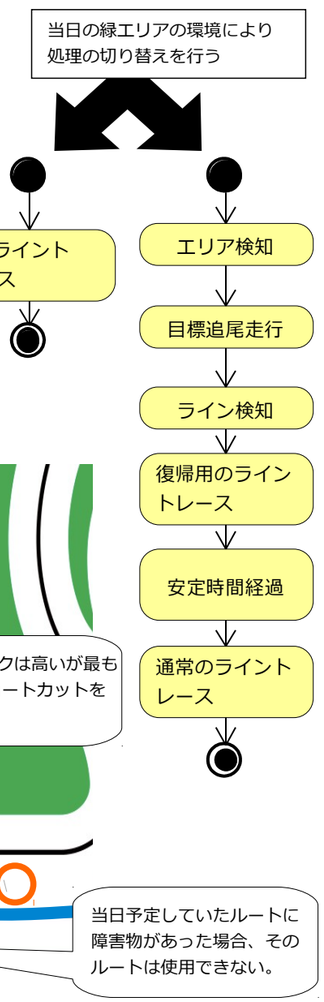




4.3 アウトコースのショートカット戦略

インコースは障害物が置かれている可能性のある緑エリアを避け、コースの白エリアが接触している箇所でショートカットを行う。基本シーケンスはインコースのショートカットと同様である。アウトコースはインコースと違い緑エリアを通過するショートカットを行うため、当日まで障害物の位置、有無が特定できない。そこで、到達目標点をいくつか用意しておき当日に切り替えることで対応する。また、用意したどの到達目標点を使用しても障害物を避けることができないと判断した場合、ショートカットは行わない。

（図はライントレースのみの処理A、緑エリアを通過する距離を極力短くする処理B、そして、最短コースである処理Cを記載）



4.4 脱線対策

ライントレース時の脱線を検知し、ラインに戻る処理を実装している。
ラインに戻る際は、ライントレースをやめた上で目標追尾走行を実施し、ラインを検知するまで目標を追尾、ラインを検知した時点で ライントレースを再開することでラインへ復帰することを基本としている。

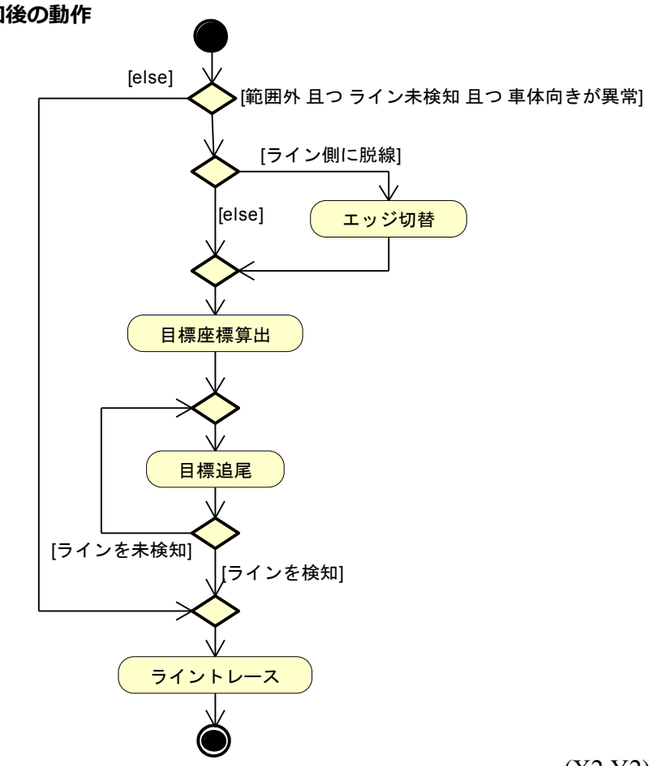
1：脱線の検知

走行体がラインを外れたことを検知する機能。
脱線は次のような条件になる。

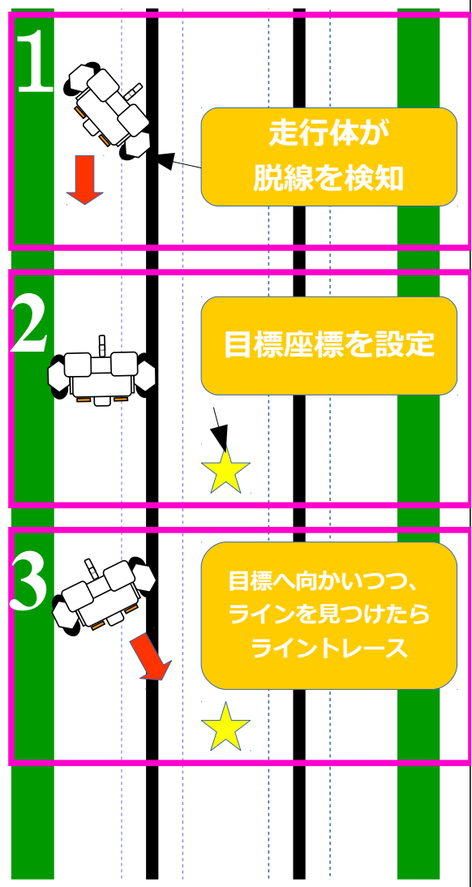
範囲外である ＆ ラインを検知していない ＆ 車体向き検知が異常値

*範囲外である： 最大でも実際のラインから10cm範囲まで離れる前に検知する

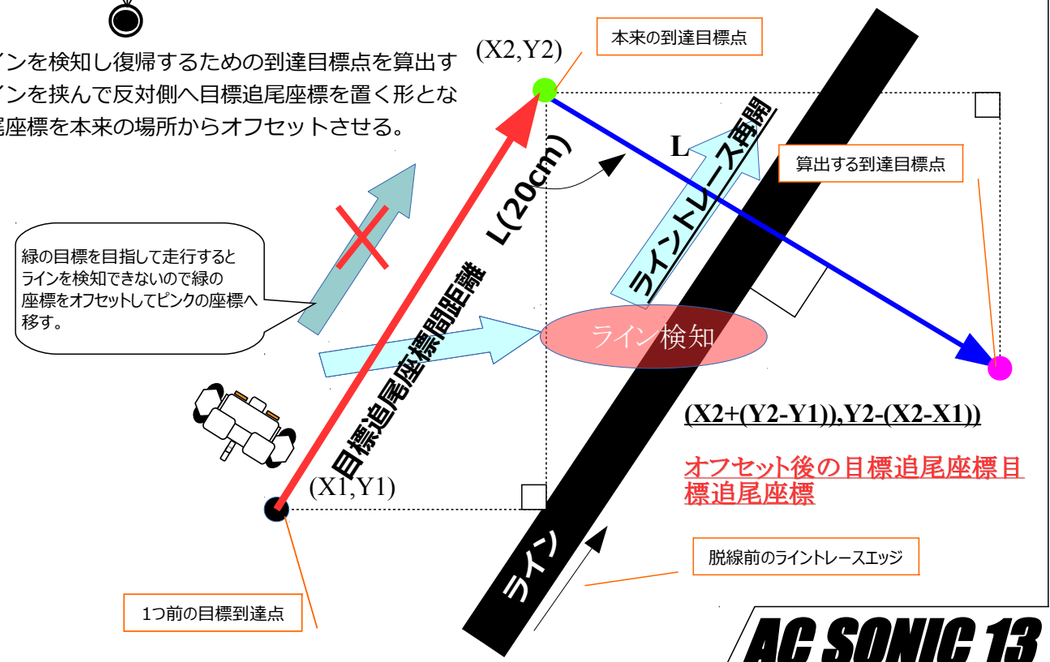
2:脱線検知後の動作



*青線は走行エリア



脱線後、確実にラインを検知し復帰するための到達目標点を算出する。走行体からラインを挟んで反対側へ目標追尾座標を置く形となるように、目標追尾座標を本来の場所からオフセットさせる。



AC SONIC 13

5. 要素技術

概要： 各要素技術の詳細を説明する。

5.1 ラインとレース動作

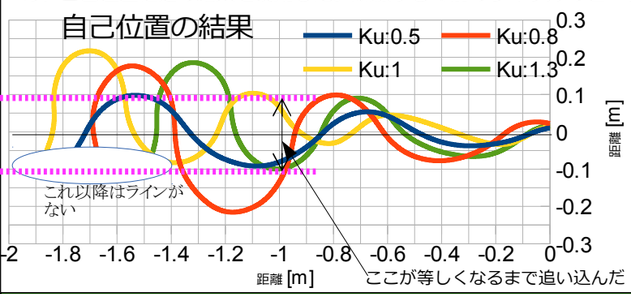
ラインとレースはPID制御を使用した。PID制御の一般的な式は以下のようになる。光センサの値と目標の輝度との差を偏差として、バランサのTURN値（旋回量)を得る。

PID制御の基本式

操作量 = **KP･e(t) + KI･∫e(t) + KD･de(t) / dt**

操作量 ：バランサの旋回値
KP：比例ゲイン　**KD**：微分ゲイン
KI：積分ゲイン　**e(t)**：偏差

PIDのパラメータは、限界感度法を用いて算出した値を元に調整を行った。調整時に予定最高速度では、長い直線用意できず、P制御のみのトレースができなかった。そこで、発散の有無を判断するために、ラインの端から出るまでのサイクルのピーク間で判断した。この時、光センサの値では飽和してしまい、サイクルのピークがわからないため、自己位置推定の結果を使用した。脱線する寸前のサイクルのピークが、0を中心に等しくなるまで追い込んだ。



ku1.3では、脱線の前は、-0.1から、+0.2になるため発散の可能性が高い。ku0.5まで下げると、脱線の前は、-0.1から+0.1となり、ぎりぎり発散しないと予想した。

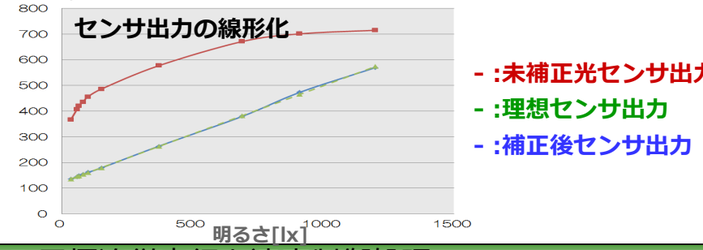
○PIDの各パラメータの係数
Kp = Ku×0.6　Ki = Tc×0.83
Kd = Tc×0.125
Ku：比例制御時の発散する限界の値
Tc：Kuの値の時の発振周期

5.2 光センサの処理～ローパスフィルタ/光センサの線形化/光センサの正規化～

光センサの外乱を防止するための方法として、ローパスフィルタと光レンジの拡張を行う。

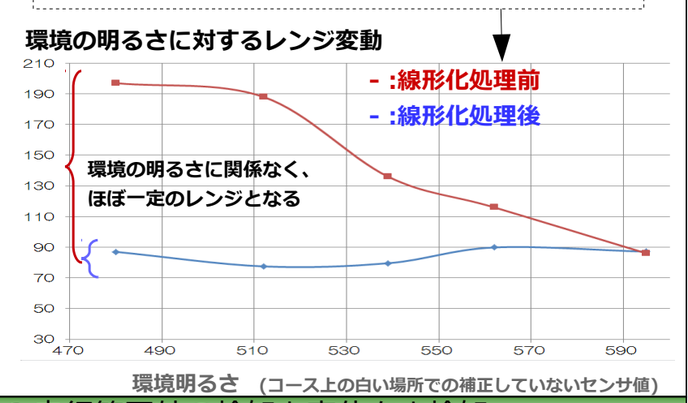
1,ローパスフィルタ：移動平均法を使ったローパスフィルタを使用することでノイズに強くなる。

2,光センサ値の線形化：　光センサ出力値は明るさに対して非線形の出力特性を持つ。走行体は環境光からの影響を補正する為、次に説明するように、白い場所でのセンサ値と黒い場所でのセンサ値を使用した正規化を行うが、環境が明るくなるほど白い場所と黒い場所での光センサ出力値の差(レンジ)が小さくなる。従って、**環境の明るさによって光センサ出力のレンジが変動しする為、環境ごとにライントレースのパラメータ等を調整する必要が出てくる。**センサ出力が明るさに対して線形であれば、環境の明るさがどのような場合でも、白い場所と黒い場所の光センサ出力値の差は大きく変化する事はなくなり、環境光に対して一定のレンジが得られるため、前述の問題を抑制できる。光センサの出力値を補正し線形特性を持たせるための補正式は明るさ毎の補正値を多次元近似で得た補正式を使用している。光センサ出力値に対して、補正式から求まる補正係数を乗算し線形補正した結果を図に示す。(**下図青線**)　補正式の次数は走行体の計算負荷を考慮し実用に問題ないような次数とした。



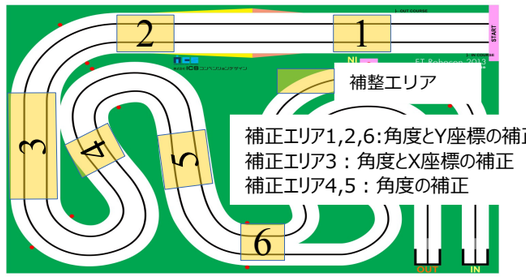
3,光センサ値の正規化：会場の照明によって、明るさが変わるため、センサの値にはオフセットがかかる。コース上の白い場所のセンサ値と黒い場所でのセンサ値を使用して正規化処理を行うことで、光のセンタ値を固定にすることができ、ライン検知の幅を固定化できる。さらに、正規化した値に係数をかけ、通常の光センサの値と同じ範囲の値にすることで、去年のプログラムを変更せず使用することができている。

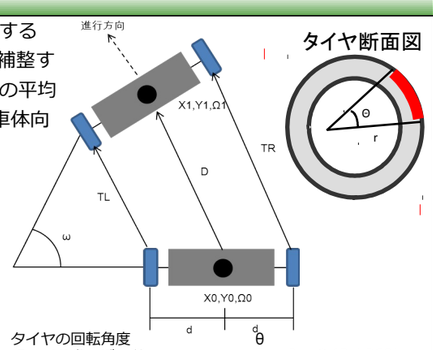
線形補正を掛けたセンサ値を施した上で、コース上白い場所と黒い場所で光センサ出力を取り、その値の差をプロットした。



5.3 自己位置推定

左右のモータのエンコーダ値から、走行体の現在の座標と向きを推定する(右式)。座標の計算値は誤差が増えていくため、補整エリアで座標を補整する。下図に補正エリアを示す。角度の補正は、エリアを通過したときの平均角度と理想の角度のずれを補整する。理想の角度の求め方は、5.5の車体向き検知の方法で算出する





		計算式
タイヤの回転角度	θ	
タイヤの進んだ距離	T = 2nr * (θ / 360)	
マシンの移動した距離	D = (TR + TL)/2	
旋回角度	ω = (TR - TL)/(2* d)	
X座標	X1 = X0 +(D * cos(Ω0 + (ω/2)))	
Y座標	Y1 = Y0 + (D* sin(Ω0 + (ω/2)))	

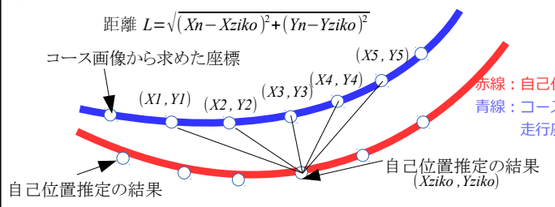
5.4 自己位置パラメータの算出

上記の自己位置推定の計算時に、タイヤ直径と車幅のパラメータを使用するが、車体の誤差によってずれてしまう。そこで、ログ取得機能で得たデータから、ホストシステムを使い以下の手順でパラメータを調整する。

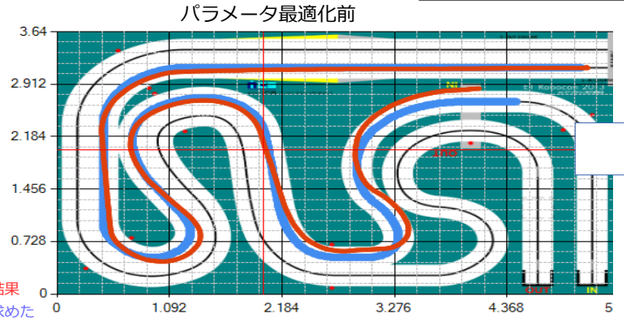
Step1：ログ取得機能を使ってライントレース走行のログデータを取得する
Step2：左右のタイヤ直径と車幅のパラメータを適当に決める
Step3：Step2のパラメータと左右のモータのエンコーダ値から、自己位置推定計算を行う
Step4：Step3の自己位置推定の座標と、コース画像から算出したライン座標の差を求める。(評価方法は下記)
Step5:Step2に戻りStep4の評価結果が最少になるように調整する。
＊上記1～5のステップはホストマシンで、自動的に調整できるようにする

＊パラメータの評価方法

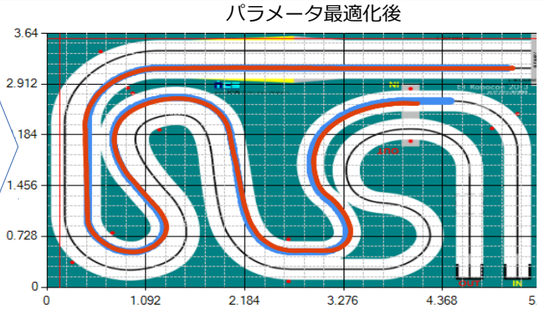
自己位置推定で求めた座標(Xziko,Yziko)から、コース画像から求めた座標(Xn,Yn)との距離Lを取得する。コース画像は1ピクセル10mmの縮尺なので、座標の間隔は10mm程度となる。座標(Xn,Yn)は(X1,Y1)から(X5,Y5)のように順次比較する。この時、座標(Xziko,Yziko)から一番近い座標(Xn,Yn)との距離Lを座標(Xziko,Yziko)のコースとの差とする。同じようにすべての自己位置推定の座標から、距離を求め、距離の最大値が、最少となるように調整する。



パラメータ最適化前



パラメータ最適化後

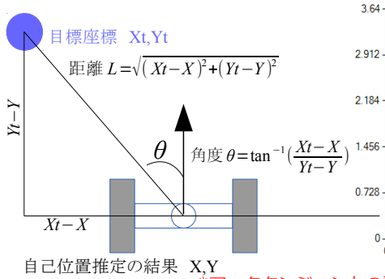


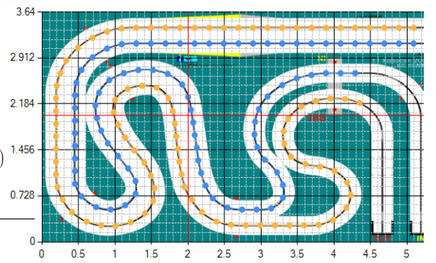
調整結果：パラメータを最適化した結果を上記に示す。ログデータは、地方/全国大会の試走会や、地方大会本番,レプリカコースで取得したデータを用いた。結果として、各会場ごとに誤差最大値の距離Lを最大値を42mm(平均14.7mm)まで小さく調整することができた。

よって、自己位置のパラメータ調整の誤差の目標値を50mmとする。

5.5 目標追従走行と速度制御処理

追尾する目標の座標をXt,Ytとし、走行体の自己位置推定の結果をX,Yとし矢印の方向を向いていたとする。この時、右の図のように、Xt､YtとX,Yの距離Lと角度θを求めることができる。目標に向かうために、距離Lと角度θを0にするように制御する。距離Lを0にするために、距離Lを偏差とし、目標値を0としてP制御を行い、角度θを偏差とし、目標値を0としてPD制御を行う。下にような目標の座標を、時間に従って切り替えていくことで、目標の追尾を行う。速度制御は、距離Lの速度制御の結果のみを使い、バランサの前進値の操作量とすることで実現する。速度の変更は、追尾動作のターゲットの更新周期を変更することで実現する





＊アークタンジェントの計算は、分母が0の時の処理や、360°以上の角度でも対応できる実装とした。5.5 走行範囲外の計算でも同様の対応とする。

5.6 走行範囲外の検知と車体向き検知

走行範囲外の検知方法のステップを以下にしめす。

Step1:最近点の算出

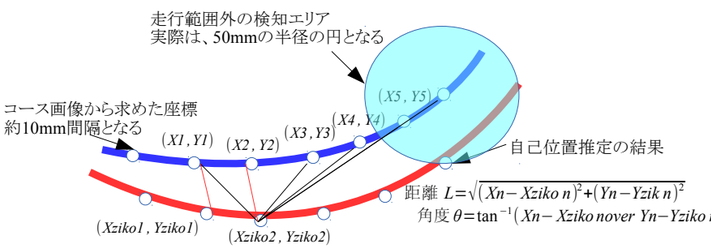
自己位置推定の結果が（Xziko1,Yziko1）の時、(X1,Y1)を最近点としている状態から、（Xziko2,Yziko2）移動したとき。現在の最近点(X1,Y1)から、(X2,Y2),(X3,Y3)…と順次距離を計算し、もっとも距離が近い点を最近点とする。コース座標の間隔は約10mm程度のため、最近点の座標は、自己位置計算の目標値の50mmから、5ポイント目まで探す。

Step2：車体向き検知

最近点と、インデックスの点のなす角を角度θの式から算出し、走行体の向きと比較し角度差が範囲外の時、走行体の向きが異常と判断する。判断する範囲は、5.4のパラメータ調整の時に、本手法と同じように角度差を求めて、そのデータの±3σを計算し、その値外を範囲外とする。

Step3：範囲外検知

最近点の座標から、距離Lが50mm以上離れた場合、範囲外と検知する



5.7 ライン検知

ライン検知では直線を走行しているときの光センサのログデータから標準偏差を求め、±3σに光が収まっているときは、ライン上にいると判断する。