1. 要求分析＆機能

AEK Runner10

本モデルのUMLはバージョン2．0に準拠しています。

■ コンセプト

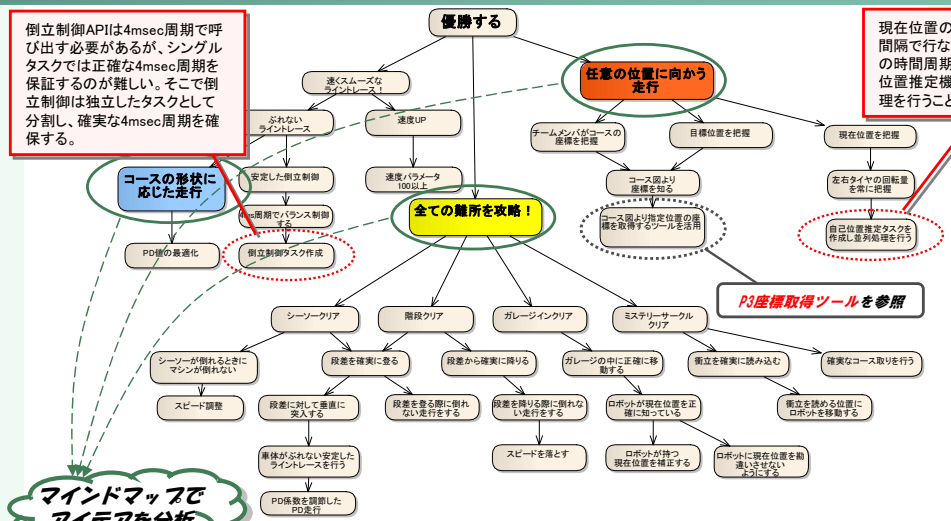
※コンセプトはマインドマップで導出（コンセプトシート参照）

①速くスムーズなライントレース！　②任意の位置に向かう走行！

③全ての難所を攻略！

● コンセプトについてゴール指向分析で要件・制約を分析

倒立制御APIは4msec周期で呼び出す必要があるが、シングルタスクでは正確な4msec周期を保証するのが難しい。そこで倒立制御は独立したタスクとして分割し、確実な4msec周期を確保する。

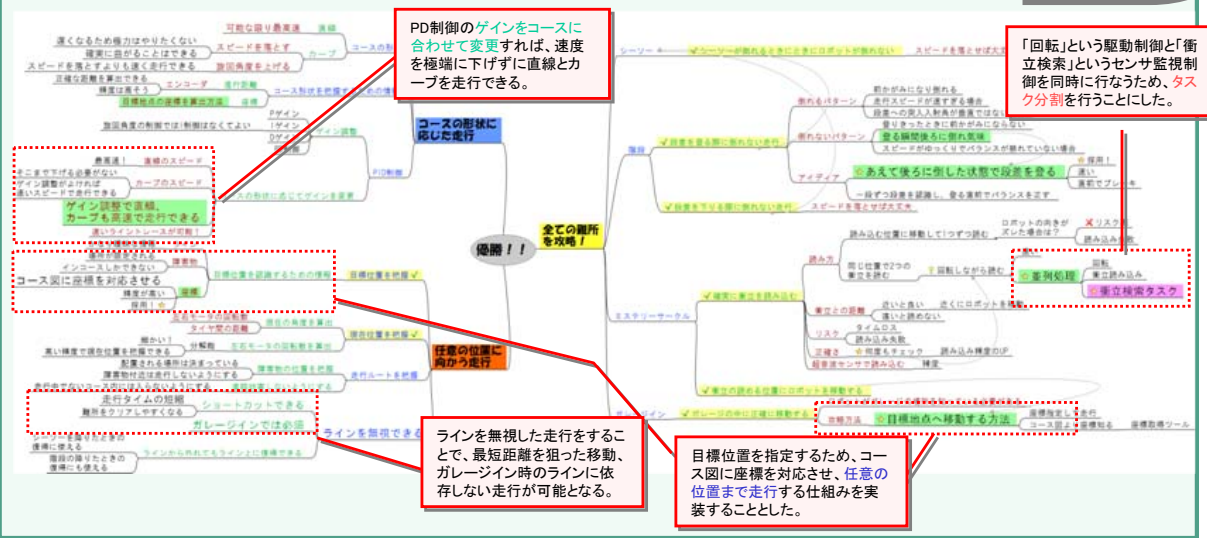


現在位置の計算処理は一定の時間間隔で行なう必要がある。この一定の時間周期を保証するため、自己位置推定機能をタスク化して並列処理を行うこととした。

マインドマップでアイデアを分析

※分岐はすべてandである。見づらさを低減するため省略した。

● ゴール指向分析で導き出した要件について、アイディアを分析



PD制御のゲインをコースに合わせて変更すれば、速度を極端に下げずに直線とカーブを走行できる。

「回転」という駆動制御と「衝立検索」というセンサ監視制御を同時に行なうため、タスク分割を行うことにした。

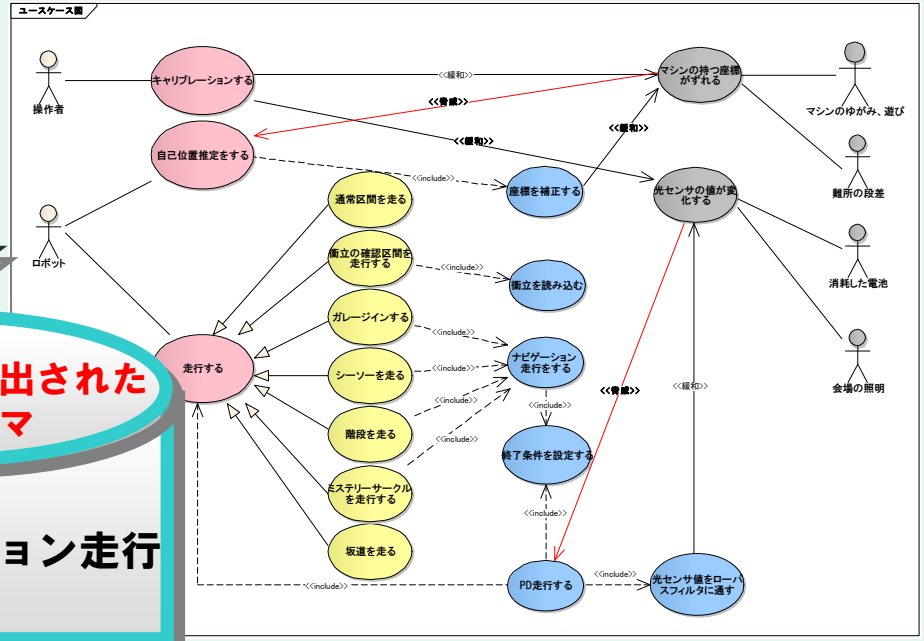
ラインを無視した走行をすることで、最短距離を狙った移動、ガレージイン時のラインに依存しない走行が可能となる。

目標位置を指定するため、コース図に座標を対応させ、任意の位置まで走行する仕組みを実装することとした。

要求分析で導き出された重要テーマ

① PD制御走行
② ナビゲーション走行
③ タスク分割

■ ユースケース図



終了条件は P4「終了条件」を参照

ユースケース記述	
名称	PD走行する
目的	ライン上をスムーズに走行する。
事前条件	マシンが走行可能な状態であること。
事後条件	終了条件を満たしていること。
基本フロー	1. 終了条件を設定する。 2. 各ゲイン、目標値を設定する。 3. 光センサの値を取得する。 4. PD演算により、旋回角度を算出する。 5. 終了条件を満たしたら走行を終了する。 6. 2.～5. を繰り返す。

ユースケース記述	
名称	ナビゲーション走行をする
目的	マシンのもつ座標をもとに走行する方向、スピードを決定し、指定した位置まで走行する。
事前条件	マシンが走行可能な状態であること。
事後条件	終了条件を満たしていること。
基本フロー	1.終了条件を設定する。 2.マシンのもつ座標を取得する。 3.取得した座標から、マシンの走行する方向、スピードを計算する。 4.計算した結果を走行パラメータに設定する。 5.終了条件を満たしたら走行を終了する。 6.2.～5. を繰り返す。

ミスユースケースの補足説明

マシンの持つ座標がずれる

脅威：
●シーソーや階段のように段差のある難所では、段差を上るときにマシンのもつ座標がずれてしまう。

●座標計算には、タイヤ半径・タイヤ間の長さという情報を用いるが、マシンのゆがみ、車軸接合部のゆがみなどが原因で計算値に誤差が生じる。

緩和：
P3 座標補正を参照
P3 キャリブレーションを参照

光センサの値が変化する

脅威：
●電池残量や会場の照明の特性によって、取得する光センサ値が異なってしまう。

●照明の光が点滅していると、取得する光センサ値が常時変動してしまう。

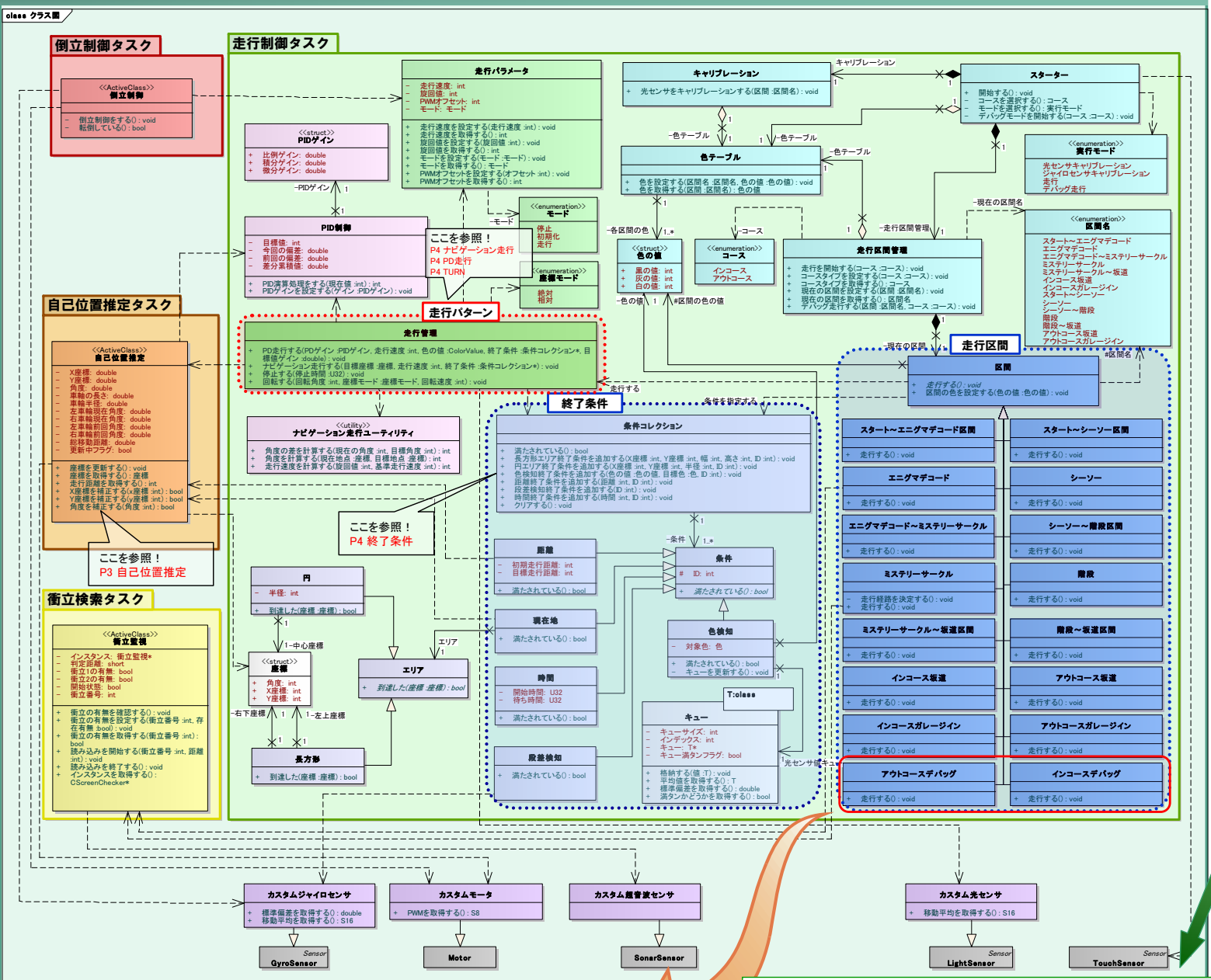
緩和：
P3 キャリブレーションを参照
P4 照明の点滅対策(移動平均処理)を参照

2. 構造＆並行性設計

AEK Runner10

本モデルのUMLはバージョン2．0に準拠しています。

■ クラス図



デバッグ走行用クラス

開発した走行パターンの挙動確認や、自己位置推定機能のキャリブレーションを行なうために、本来の区間とは独立した走行区間クラスを用意している。
これにより本来の走行用プログラムに影響を与えることなく、プログラムのデバッグが可能となった！

◆ AEK Runnerの排他制御

nxtOSEKのリソース機能を使用することで、排他制御を実現している。「走行制御タスク」が走行パラメータクラスにパラメータを設定する間は、リソースを取得し他のタスクからの割り込みが発生しないようにしている。このことにより、パラメータ郡を設定中に割り込みが発生し中途半端にパラメータが設定された状態になる「データの泣き別れ」を発生させることなく動作させることができた。

■ 並行性

● タスク分割

本プログラムでは、以下の4つのタスクを使用している。

優先度 高

倒立制御タスク 4msec周期起動(実行時間：1msec以下)
指定されたスピードと旋回角で走行するためのモータPWMを決定する

自己位置推定タスク 10msec周期起動(実行時間：1msec以下)
モータの回転角度をもとに、コース上におけるマシンのXY座標と角度を計算する

衝立検索タスク 50msec周期起動(実行時間：1msec以下)
超音波センサの情報をもとに衝立の有無を確認する

走行制御タスク 常時起動
現在選択している走行方法に基づいて、走行スピードと旋回値を決定する

優先度 低

※各タスクの実行時間は、タスク起動時のシステム時間ータスク終了時のシステム時間を計測した結果である。測定値は1ms以下であった。

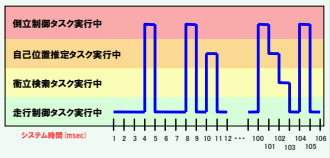
また、上記のタスクの起動周期は以下の理由から決定している。

倒立制御タスク	倒立制御APIが4msec周期の呼び出しを前提としているため4msecとしている。
自己位置推定タスク	マシンが高速で走行している状態でも、マシンが特定の位置にいることを把握するため、小さな移動距離を検出する必要がある。そこで本チームでは、1回の座標更新に伴う最大の座標更新量を20mm以下となるように定めている。その要件を満たすため、起動周期を10msecとしている。 ※走行速度120で走行させたとき、10msecあたりの移動距離は20mm以下となることを確認している。
衝立検索タスク	衝立の有無の確認は、衝立の前でマシンを90°回転させながら行なっている。その際、5°単位で衝立有無の確認を行なえば、確実な衝立読み込みを実現可能と判断したため、起動周期を50msecとしている。 ※2秒間かけてマシン90°回転させたとき、衝立読み込み間隔は5°以下となることを確認している。

● タスクスケジューリング

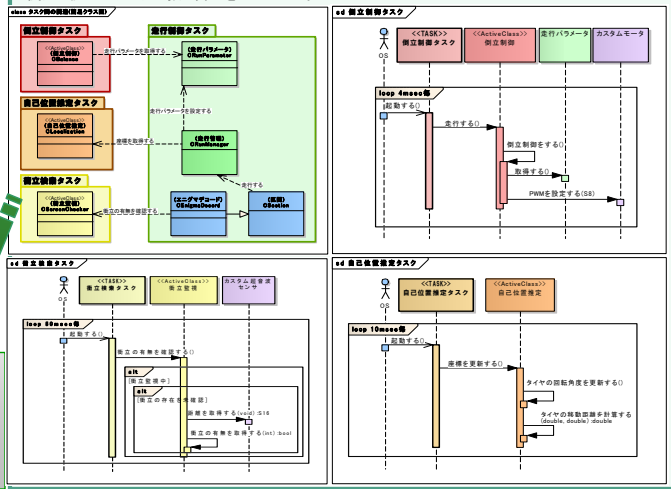
右図は、各タスクのスケジューリングについて示している。「走行制御タスク」は常時起動し、OSからタスクが呼び出された際により高い優先度のタスクが実行される(※)。同時間に複数タスクが呼び出された場合、優先度の高いタスクから順次実行される。各タスクの実行時間が1msec以下のため、全タスクが同時に動いても倒立制御タスクの周期を乱すことは無い。

※OILファイルの設定でプリエンティブ(SCHEDULE=FULL)と設定した。



● タスク間通信

共有メモリを介した各タスク間のデータのやり取りを行うことでタスク間の通信を実現している。走行制御タスクが、走行制御タスクと各タスクとの共通パラメータの設定や取得を行っている。下図はタスク間の関連と個々のタスクの振る舞いを示している。

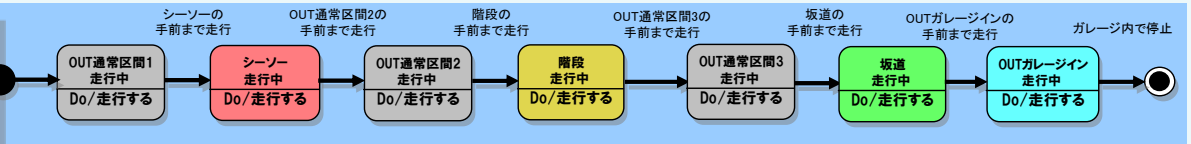


3. 振る舞い ～コース戦略・基本戦術～

AEK Runner10

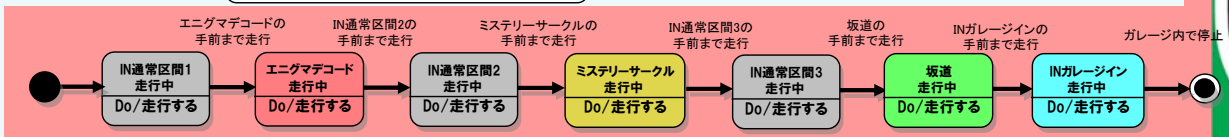
本モデルのUMLはバージョン2．0に準拠しています。

■ 状態マシン図（走行区間の切替）



難所の詳細はP5参照

※走行区間の名称を 一部省略して記述している。
例:「スタート～エニグマデコード区間」⇒「IN通常区間 1」



■ 自己位置推定

● 概要

左右のモータのエンコーダ値から、マシンが現在走行している座標(下記参照)を計算する。

● 特長

座標を利用することで、以下の動作が可能となる。

・指定した座標への走行

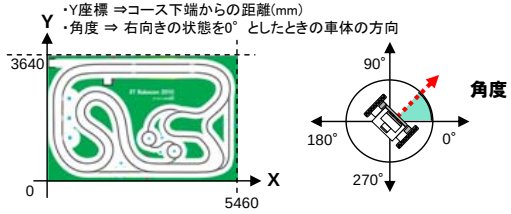
P4「ナビゲーション走行」参照

・任意の座標での走行方法の切り替え

P4「終了条件」参照

座標について

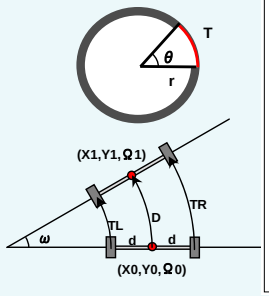
本モデルでは、以下の3つの情報をまとめて「座標」としている。
・X座標 ⇒コース左端からの距離(mm)
・Y座標 ⇒コース下端からの距離(mm)
・角度 ⇒ 右向きの状態を0°としたときの車体の方向

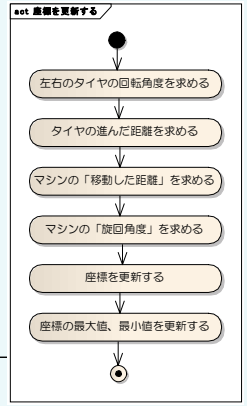


● 振る舞い

「自己位置推定タスク」で座標を更新する。座標の更新は以下の手順で行う。

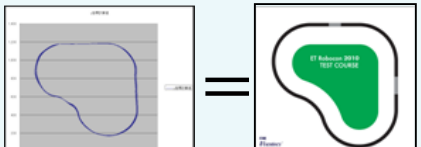
1. 左右のタイヤの回転角度を求める。
   現在のエンコーダ値E1と、前回計算時に取得したエンコーダ値E0から、タイヤの回転角度θを取得する。
   【タイヤの回転角度】θ =E1−E0
2. タイヤの進んだ距離を求める。
   【タイヤの進んだ距離】T=2πr*(θ/360)（r:タイヤ半径）
   ※左右のタイヤの進んだ距離をそれぞれTL、TRとする。
3. マシンの「移動した距離」を求める。
   【移動した距離】D =(TR+TL)/2
4. マシンの「旋回角度」を求める。
   【旋回角度】ω =(TR−TL)/(2*d)（d:車軸長）
5. 座標を更新する。
   ※更新前と更新後の座標をそれぞれ(X0,Y0,Ω0)、(X1,Y1,Ω1)とする。
   【X座標】X1 =X0+(D*cos(Ω0+(ω/2)))
   【Y座標】Y1 =Y0+(D*sin(Ω0+(ω/2)))
   【角度】Ω1=Ω0+ω
6. 座標の最大値、最小値を更新する。





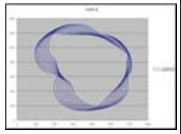
● 検証結果

テストコースで数回周回させたときの座標計算結果を取得し、走行中に計算した座標が実際のコースと一致することを確認した。



計算誤差の緩和

右の図は、車軸長を実際の長さより1mm小さい値に設定した場合の座標計算結果であるである。図からわかるように、「車軸長」、「タイヤ半径」の設定誤差は座標計算に大きな影響を及ぼす。この影響を最小限に抑えるため、**「自己位置推定のキャリブレーション」**を実施する。
また、走行中には「タイヤのスリップ」や「傾斜部・段差における計算誤差」も発生する。これらの影響を抑えるため**「座標補正」**機能を実装している。



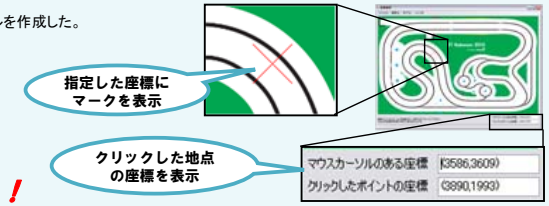
■ 補足機能

● 座標取得ツール

実装作業、デバッグ作業における座標の設定作業を容易にするため、以下の機能を持つツールを作成した。
○ コース上をクリックすることで、該当箇所の座標を取得
○ 座標を入力することで、コース上の該当箇所にマークを表示
このツールを使用することで、コース上の位置と座標を視覚的に結びつけることができた。

① スタート地点の座標は？
⇒クリックしたらすぐ分かる！
② X座標 **3000**、Y座標**1400** はコースのどの辺？
⇒マークが出てすぐ分かる！

★座標の設定値が正確になり、走行性能アップ！！



● キャリブレーション

以下の2つの機能においてキャリブレーションを行っている。
○ 光センサ
光センサから取得する値は環境光の影響により異なる。そのため、実環境におけるライン上(白/黒/マーカ)の光センサ値を取得し、ライントレースに使用する閾値を設定しなおす。
○ 自己位置推定
座標計算に使用する車軸長とタイヤ半径は、マシンの状態によって変動する。そこで、コース走行時の座標計算結果をログから取得し、実際のコース座標と一致するように車軸長とタイヤ半径の設定パラメータを調整する。

★マシンや環境が変わっても大丈夫！！

● 座標補正

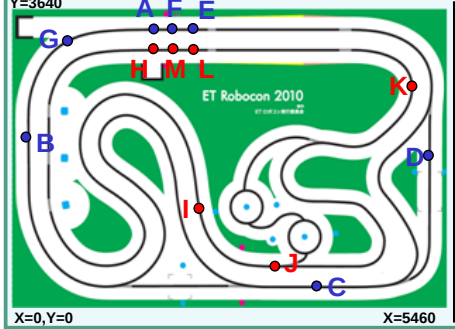
● 概要

コース走行中に発生する座標誤差を、補正ポイント(下記参照)走行時に補正する。

● 振る舞い

座標の補正は以下の手順で行う。
1. 補正ポイントを通過する。
2. 補正ポイントに対応する補正値(下表参照)を座標に設定する。

補正ポイント一覧



補正値対応表

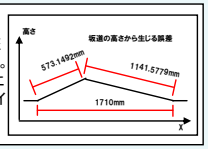
	補正ポイント	補正値 角度	X	Y
OUT	A	180	-	3405
	B	270	177	-
	C	0	-	241
	D	90	5239	-
	E	-	+5	-
	F	180	-	3402
	G	-	718	-
IN	H	180	-	3128
	I	-	2316	-
	J	0	-	513
	K	-	5026	-
	L	-	+5	-
	M	180	-	3128

補正ポイントの選定基準

補正ポイントは、X座標、Y座標、角度の内、いずれかの値が特定可能となる地点を選定している。具体的には、以下の条件を満たす地点を補正ポイントとしている。
・コース内の水平方向、垂直方向の直線
・ライン上でX座標、Y座標が極大、極小となる地点
【例】下図のような直線を走行した場合、Y座標と角度(180度)を特定することが可能となる。

坂道における誤差

坂道を走行すると、計算上の座標と実座標では、X座標に約5mmの誤差が生じる。この誤差は直後のガレージインの走行に影響するため、坂道の走行後の補正ポイント(左図E、L)で補正を行う。



★座標の誤差を補正できる！！

4. 振る舞い ～要素技術・戦術～

AEK Runner10

本モデルのUMLはバージョン2．0に準拠しています。

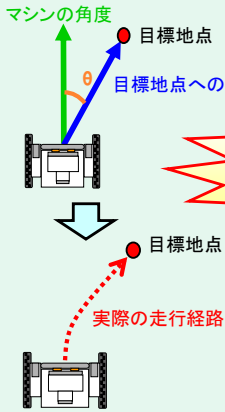
■ ナビゲーション走行

● 概要

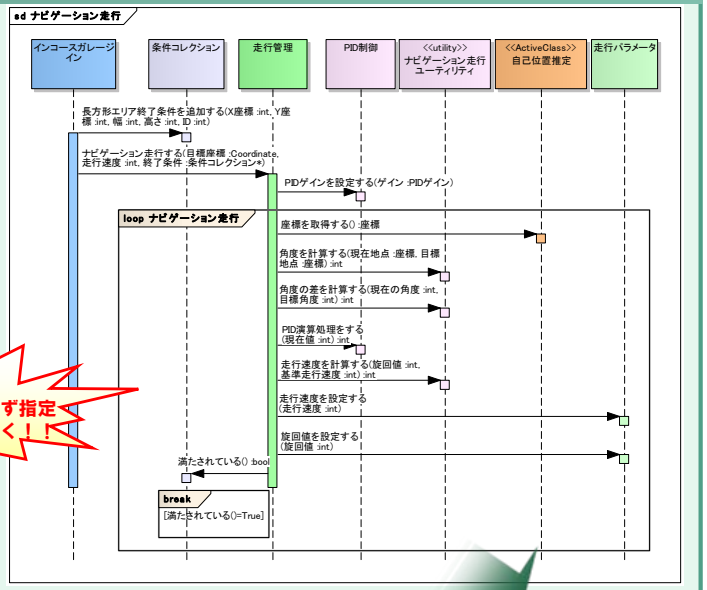
マシンの持つ座標情報をもとにして、指定された座標（目標地点）に向かって走行する。
例として、右図にインコースガレージイン走行時の動作を示す。

● 走行方法

マシンの現在の角度と目標地点への角度の差θが0になるように旋回値を決定する。また、θを0にする旋回値はPD制御によって計算している。



どの位置にいても必ず指定した座標にたどり着く！！



ここに使える！
・ガレージインクリア
・ミステリーサークルクリア

■ PD走行

● 概要

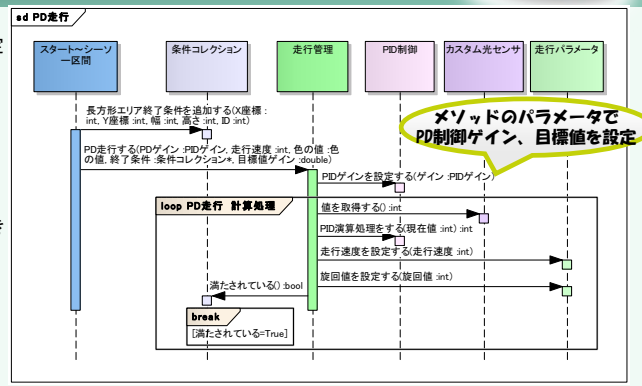
光センサから得られる値からPD制御を行い、旋回値を決定して走行する。
例として、右図にスタート～シーソー区間走行時の動作を示す。

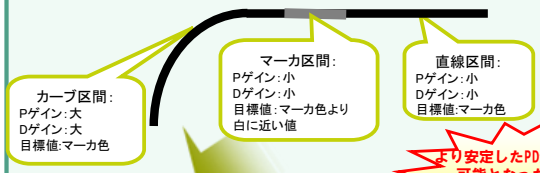
● 走行方法

現在の光センサの値と、光センサの目標値をもとにしてPD制御による計算を行ない、旋回値を決定する。
また、カーブ、直線、難所 すべての区間において、ふらつきの無いスムーズな走行を実現するため、以下のような仕組みを採用している。

・動的な目標値の設定
光センサの目標値を可変とすることで、色が異なるマーカ付近や難所もぶれることなく走行！

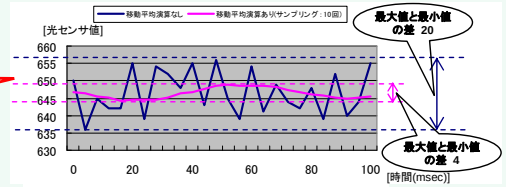
・動的なPDゲインの設定
PDゲインを変更することによって直線、カーブとも最適なPDゲインで走行！





■照明の点滅対策(移動平均処理)

下のグラフは照明が点滅している環境下(水銀灯)で、マシンを同じ位置で固定させて光センサ値を取得したときのものであり、取得値がばらついた結果になっている。
対策として取得した光センサ値の移動平均をとることで、照明の点滅による急激な値変化の影響を緩和している。



※移動平均時のサンプリング個数については、照明機器が不明なため大会会場で調整を行う。

より安定したPD走行が可能となった！

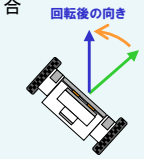
ここに使える！
・ラインのある全ての区間
・直線区間での座標補正

走行パターンの二本柱である「ナビゲーション走行」と「PD走行」！
これに終了条件を組み合わせることであらゆる区間を走破する！！

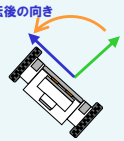
■ TURN

マシンを指定した角度に回転させる。
角度の指定方法には、絶対角度指定と相対角度指定がある。

絶対角度90°で回転した場合



相対角度90°で回転した場合

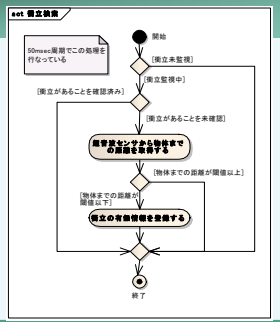


ここに使える！
・衝立の検索
・ミステリーサークル
・ガレージイン

■ 衝立検索

超音波センサを利用した衝立の確認は、独立したタスクにおいて50msec間隔でチェックしている。
右に衝立チェックのアクティビティ図を示す。

※衝立検索は、インコース走行時のみ行う。



ここに使える！
・衝立の検索

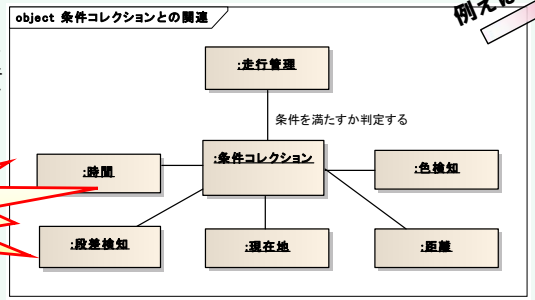
■ 終了条件

ナビゲーション走行、PD走行を終了するタイミングは、終了条件を指定することで制御する。
本プログラムでは、以下の5つの終了条件を用意している。

終了条件名	内容	判定に使用する情報
エリア	特定の範囲内のエリアに進入したと判定した際に終了する。 長方形エリアと円エリアを指定可能。	座標情報（X,Y座標）
距離	特定の距離を走行したと判定した際に終了する。	走行距離
時間	特定の時間が経過したと判定した際に終了する。	システム時間
色検知	特定の色のコースを走行したと判定した際に終了する。	光センサ値
段差検知	マシンが段差にぶつかったと判定した際に終了する。	ジャイロセンサ値

● 条件コレクション

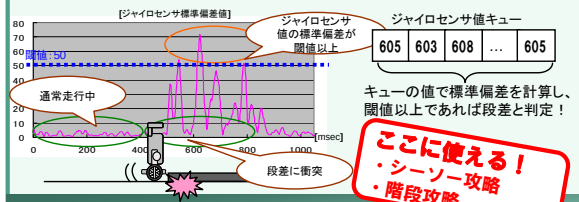
上記の終了条件を複数個指定して、各走行メソッドのパラメータとして渡すことで、いずれか1つの条件が満たされたときに走行が終了する。この終了条件を格納するのが「条件コレクション」クラスである。右は条件コレクションと各終了条件の関連を示すオブジェクト図である。



様々な終了条件を用意することで、戦略の幅が大幅に拡大！！

■ 段差検知（終了条件）

マシンが段差にぶつかる際に車体がふらつきジャイロセンサ値が大きく変化することを利用して、段差検知を行う。ジャイロセンサ値の変化の検出には、標準偏差をとることで差分を検知しやすくしている。



ここに使える！
・シーソー攻略
・階段攻略

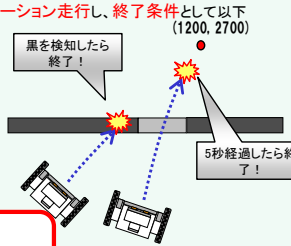
■ 終了条件使用例

例えば・・・

座標（1200,2700）を目標地点としてナビゲーション走行し、終了条件として以下の2つを指定する。
・色検知終了条件（黒を検知）
・時間終了条件（5秒）

この場合、黒を検知したら現在の走行を終了、黒を検知できなくても5秒経過すれば現在の走行を終了する。

さらに、黒を検知したのか、5秒経過したのかによって、処理を分岐させることが可能である。



こんなメリットが！
・１つの条件に頼らない保険効果！
・自由な組み合わせによる柔軟な戦略！

P5 シーソーを参照！

5. 振る舞い ～難所攻略～

AEK Runner10

本モデルのUMLはバージョン2．0に準拠しています。

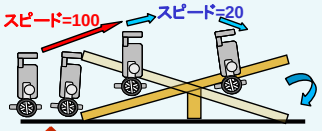
OUT

IN

シーソー

振る舞い

PD走行を行い、スピードを調整しながら攻略！
シーソーが反転する直前にスピードを下げて走行！
段差から降りた後はナビゲーション走行でライン復帰！



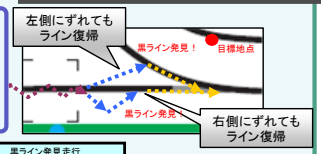
リスク

段差から降りるとき車体がふらついているとラインから離れた場所に降りる可能性あり

これで解決

段差を降りた後、黒検知のナビゲーション走行を行うことで段差を降りた地点がずれても、確実なライン復帰が可能！

※階段でも同様のロジックを採用



ブロック図

シーソー前まで走行
走行:PD走行
スピード:130
PD係数:P=0.4 D=15.0
終了条件:段差検知

→ シーソー登り走行
走行:PD走行
スピード:50
PD係数:P=0.2 D=10.0
終了条件:走行距離(130)

→ シーソー中盤走行
走行:PD走行
スピード:20
PD係数:P=0.2 D=10.0
終了条件:走行距離(110)

→ シーソー下り走行
走行:PD走行
スピード:20
PD係数:P=0.2 D=10.0
終了条件:走行距離(200)

→ 黒ライン発見走行
走行:ナビゲーション走行
スピード:50
目標値:X=3675,Y=888,
終了条件:色検知(黒)

階段

振る舞い

終了条件に走行距離を設定してPD走行。スピードを調節しながら段差を登る！
段差登板時は、PWMオフセットを設定して走行！
PWMオフセット値を加算することによりタイヤの回転速度が通常よりも増加して車体が後傾になる。
車体を後傾にさせることで段差を上る際に車体が前かがみになる力を相殺する！

リスク

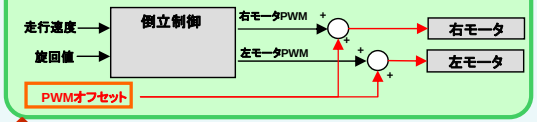
高速で段差を登ろうとすると車体が前傾になりバランスを崩す可能性大！

これで解決

段差を登る際、モータのPWMオフセットをプラス方向にセットして後傾姿勢を意図的に生成。コントロールされたアンバランスにより階段を走破する！

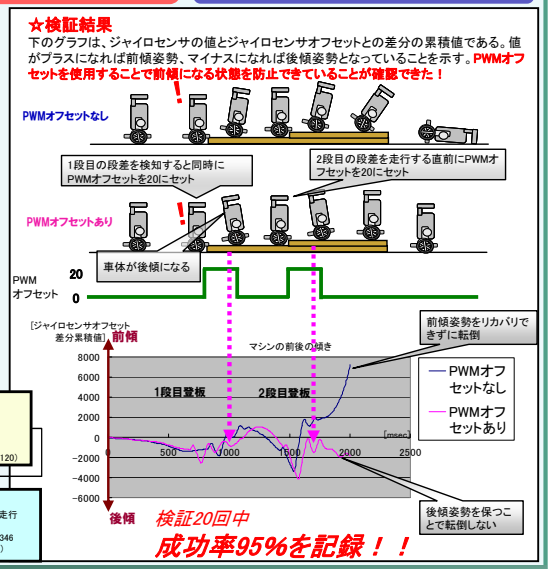
☆工夫ポイント　PWMオフセット

通常、各モータのPWM（Pulse Width Modulation）には倒立制御関数によって得られた値を設定するが、本プログラムではこの値に対してさらにオフセット値を加算している。ここでは、この値をPWMオフセットと呼んでいる。PWMオフセット値を変更することにより、意図的にマシンの姿勢を変化させることを可能としている。



☆検証結果

下のグラフは、ジャイロセンサの値とジャイロセンサオフセットとの差分の累積値である。値がプラスになれば前傾姿勢、マイナスになれば後傾姿勢となっていることを示す。PWMオフセットを使用することで前傾になる状態を防止できていることが確認できた！



検証20回中

成功率95%を記録！！

ブロック図

階段前まで走行
走行:PD走行
スピード:120
PD係数:P=0.1 D=0.0
PWMオフセット=0
終了条件:段差検知

→ 1段目を登る
走行:PD走行
スピード:150
PD係数:P=0.1 D=0.0
PWMオフセット=20
終了条件:走行距離(50)

→ 2段目の手前まで走行
走行:PD走行
スピード:150
PD係数:P=0.1 D=0.0
PWMオフセット=0
終了条件:走行距離(100)

→ 2段目を登る
走行:PD走行
スピード:70
PD係数:P=0.2 D=0.0
PWMオフセット=20
終了条件:走行距離(120)

→ 2段目真ん中付近まで走行
走行:PD走行
スピード:30
PD係数:P=0.2 D=0.0
PWMオフセット=20
終了条件:走行距離(50)

→ くだりの段差前まで走行
走行:PD走行
スピード:70
PD係数:P=0.8 D=40.0
PWMオフセット=0
終了条件:走行距離(150)

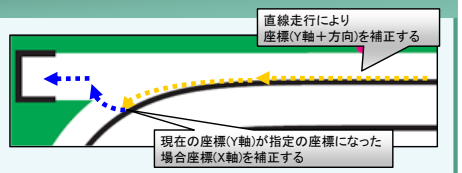
→ マーカの直前まで走行
走行:PD走行
スピード:100
PD係数:P=0.4 D=20.0
PWMオフセット=0
終了条件:座標(3232,2251)

→ 黒ライン発見走行
走行:ナビゲーション走行
スピード:50
目標値:X=4990,Y=2346
終了条件:色検知(黒)

OUTガレージイン

振る舞い

①終了条件にガレージ近くのライン上へ座標を設定し、PD走行でライントレース！
②座標に到達すると、終了条件にガレージの座標を設定しナビゲーション走行！
③ガレージ内に入ると走行停止！



☆工夫ポイント

直線の走行による座標補正と、カーブを利用した座標補正。ギリギリまでライントレースを行うことで、ナビゲーション走行時の座標誤差を低減。
だから出来る確実なガレージイン！！

ブロック図

直線走行
走行:PD走行
スピード:160
PD係数:P=0.4 D=20.0
終了条件:座標(1400,3392)

→ カーブを走行
走行:PD走行
スピード:140
PD係数:P=0.6 D=20.0
終了条件:座標(718,3277)

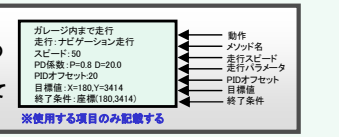
→ ガレージ付近まで走行
走行:ナビゲーション走行
スピード:60
目標値:X=391,Y=3414
終了条件:座標(391,3414)

→ ガレージ内まで走行
走行:ナビゲーション走行
スピード:20
目標値:X=180,Y=3414
終了条件:座標(180,3414)

エニグマデコード

ブロック図

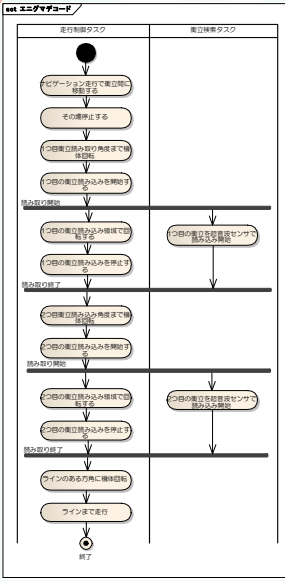
右図をブロック図と命名する。
各オブジェクト内のメソッドやパラメータの設定をブロック図にて表現！
このブロック図を組み合わせる事によって処理の手順を明記！



※使用する項目のみ記載する

振る舞い

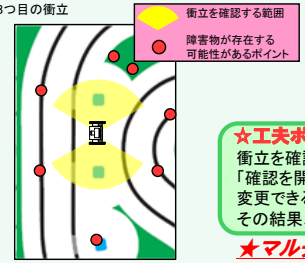
ナビゲーション走行によって1つ目と2つ目の衝立間に移動！
その場で回転しながら2つの衝立の有無を確認する。衝立を読み取った後は、下図右のラインまで進みショートカットを行う。

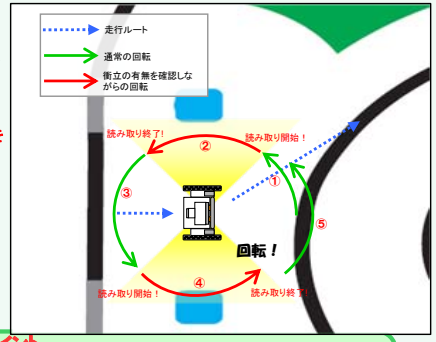


確認範囲

衝立を確認する範囲は、角度:120度、距離:45cmとした(下図参照)。これは、障害物(※)の誤検知を防ぐために読み込み距離を小さくし、衝立を見落さないために角度範囲を大きくした結果得られた値である。これにより、**安全で精度の高い衝立確認を行えることを確認した。**

※障害物とは以下を指す。
・コースの緑のエリアに置かれるオブジェクト
・アウトコースを走行するマシン
・3つ目の衝立





☆工夫ポイント

衝立を確認する処理(衝立検知)を別タスクに分割したことにより、「確認を開始／終了するタイミング」と「確認する距離」を簡単に変更できるようになった。
その結果、範囲の調整を簡単に行うことができた。

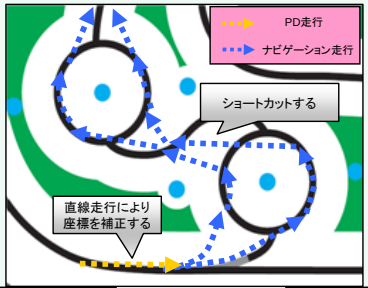
★マルチタスクのメリットを活かして走行性能アップ！！

ミステリーサークル

振る舞い

エニグマデコードで読み取った衝立情報を基に走行ルートを決定！PD走行で座標を補正した後、ナビゲーション走行で走破！

解読パターン (衝立1個目,2個目の有無)	走行ルート
パターン1 (なし,なし)	→
パターン2 (なし,あり)	→
パターン3 (あり,なし)	→
パターン4 (あり,あり)	→



リスク

ナビゲーション走行でゲートを通過するため、ゲートに衝突する可能性あり(ゲートの幅が狭い)

これで解決

ミステリーサークルの入り口で座標を補正してナビゲーション走行の精度アップ！

★ナビゲーション走行でショートカット！！！

ブロック図

座標を補正する
走行:PD走行
スピード:30
PD係数:P=0.8 D=20.0
終了条件:座標(3540,1030)

→ 1つ目の左側ゲートまで走行
走行:ナビゲーション走行
スピード:30
目標値:X=3746,Y=1222
終了条件:座標(3746,1222)

→ 1つ目の左側ゲートを走行
走行:ナビゲーション走行
スピード:70
目標値:X=3490,Y=966
終了条件:座標(3490,966)

→ 合流地点まで走行
走行:ナビゲーション走行
スピード:70
目標値:X=3298,Y=1052
終了条件:座標(3298,1052)

→ 2つ目の左側ゲートまで走行
走行:ナビゲーション走行
スピード:70
目標値:X=2920,Y=1052
終了条件:座標(2920,1052)

→ 2つ目の左側ゲートを走行
走行:ナビゲーション走行
スピード:70
目標値:X=2758,Y=1350
終了条件:座標(2758,1350)

→ 出口まで走行
走行:ナビゲーション走行
スピード:70
目標値:X=2914,Y=1585
終了条件:座標(2914,1585)

1つ目の右側ゲートまで走行
走行:ナビゲーション走行
スピード:70
目標値:X=3803,Y=739
終了条件:座標(3803,739)

→ 1つ目の右側ゲートを走行
走行:ナビゲーション走行
スピード:70
目標値:X=3540,Y=1030
終了条件:座標(3540,1030)

→ 2つ目の右側ゲートまで走行
走行:ナビゲーション走行
スピード:70
目標値:X=3121,Y=1137
終了条件:座標(3121,1137)

→ 2つ目の右側ゲートを走行
走行:ナビゲーション走行
スピード:70
目標値:X=3096,Y=1306
終了条件:座標(3096,1306)

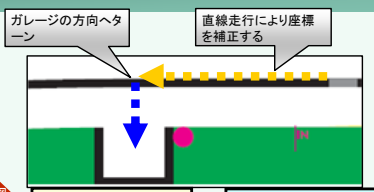
INガレージイン

振る舞い

①終了条件にガレージ近くの座標を設定し、PD走行でライントレース！
②座標に到達後、終了条件にガレージ内の座標を設定してナビゲーション走行！
③ガレージに進入後、走行停止！

☆工夫ポイント

以下の処置によりナビゲーション走行の精度を上げている。
・直線走行時のY軸・方向の座標補正
・坂道前カーブ走行時のX軸の座標補正
・坂道走行によって生じる座標の誤差補正
（坂道前カーブの座標補正、坂道の段差の誤差修正についてはP3座標補正参照）
だから出来る確実なガレージイン！！



ブロック図

ガレージ付近まで走行
走行:PD走行
スピード:120
PD係数:P=0.4 D=5.0
終了条件:座標(1759,3152)

→ ガレージ内まで走行
走行:ナビゲーション走行
スピード:50
目標値:X=1741,Y=2860
終了条件:座標(1741,2860)