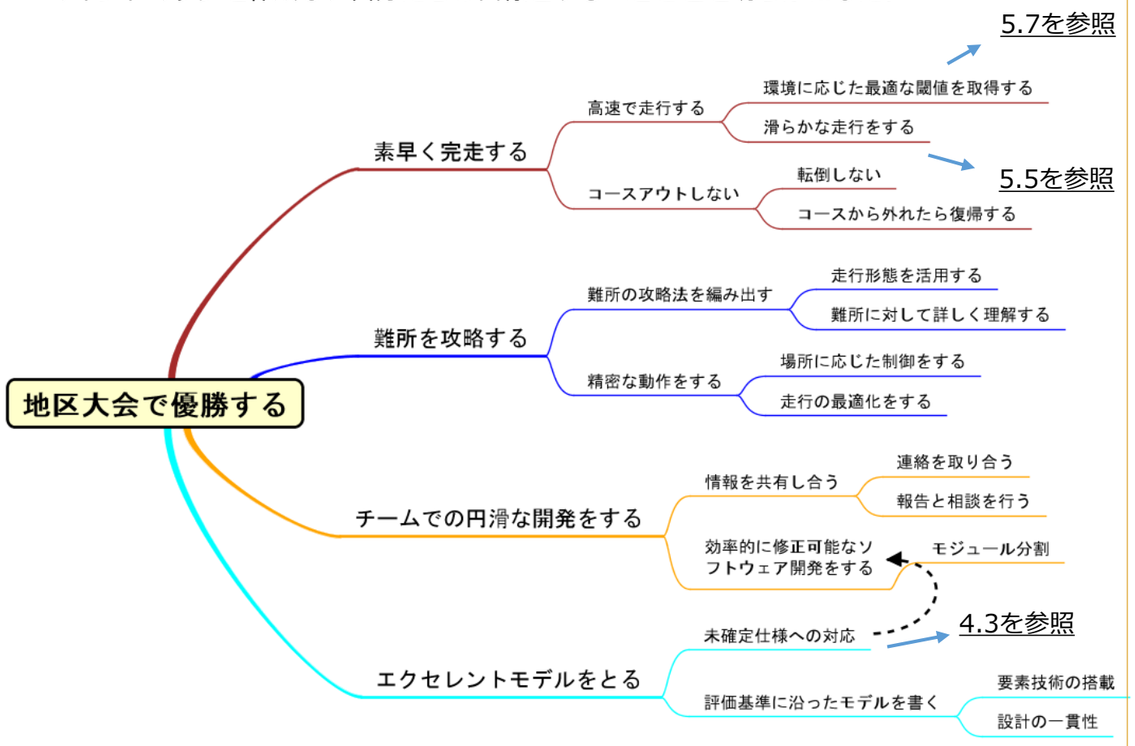
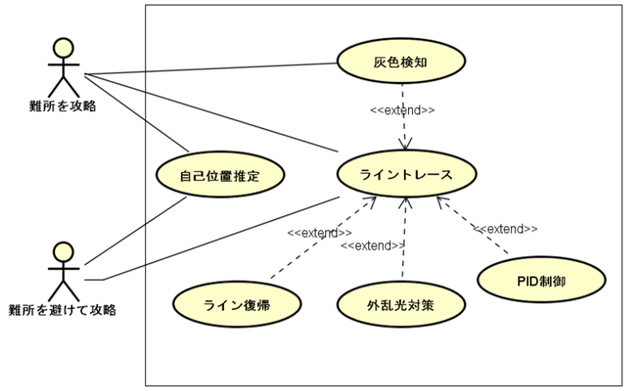
1.1マインドマップ

マインドマップを作成し、自分たちの目標とするべきことを明らかにした。

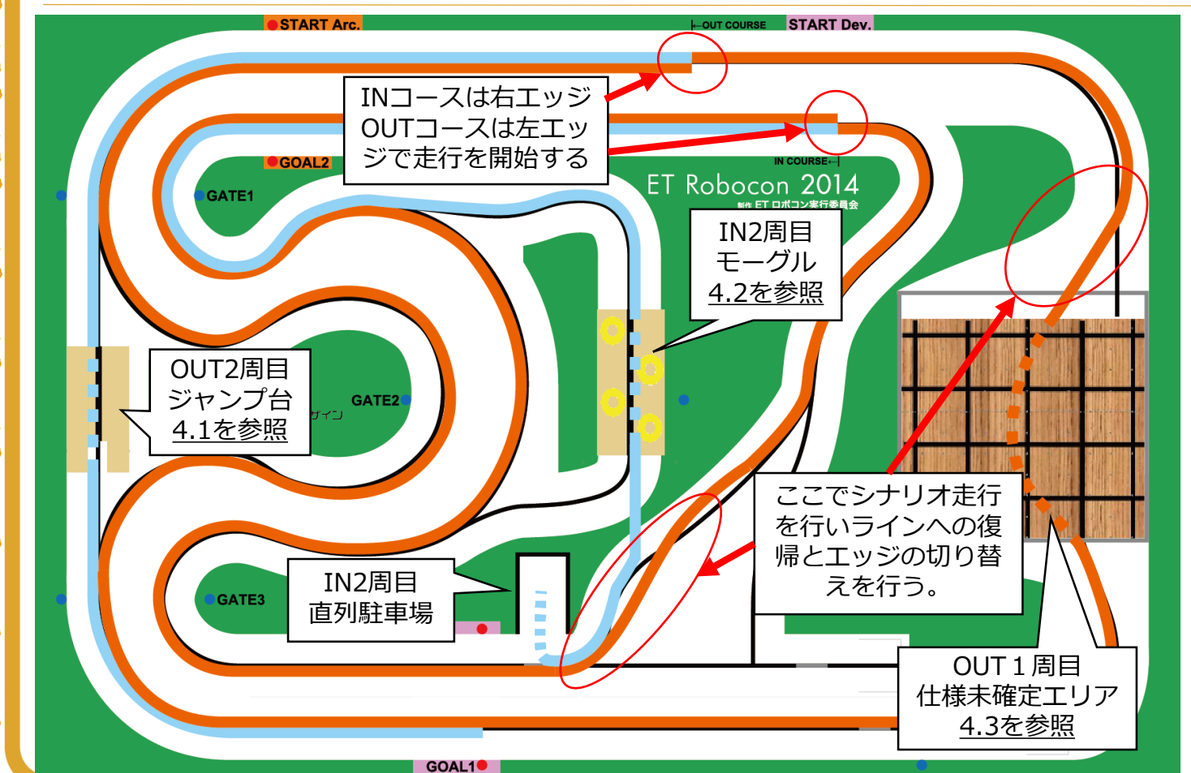


1.2ユースケースと記述

走行に関するユースケースを作成し、
難所を避けて攻略する方法、難所を攻略する方法の走行方法を明らかにした。
(攻略する難所スルーする難所に関しては1.4走行スケジュールを参照)

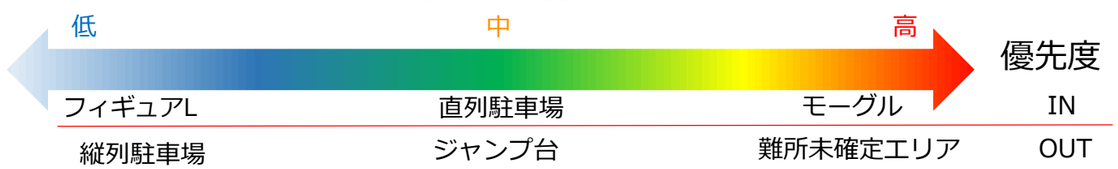


項目	内容
ユースケース	ライントレース
概要	走行体をラインに沿って走行させる
アクター	障害物を攻略 障害物を避けて攻略
事前条件	走行体が走行していること
事後条件	走行体がラインに沿って走行していること
基本系列	1.走行体（システム）は走行している 2.走行体（システム）は現在位置を取得する 3.走行体（システム）はライトセンサにより走っている地点の色を取得 4.走行体（システム）はPID制御により走行体の走る向きを調整する 5.走行体（システム）は走行を終了する 6.本ユースケースを終了する
代替系列	Alt-1.基本系列2において走行体が難所に差し掛かった場合 1.走行体（システム）は必要に応じてライントレースを中断する 2.走行体（システム）は難所を攻略する 3.基本系列2に進む Alt-2.基本系列3で灰色を検知した場合 1.走行体（システム）は現在位置により難所であるかを判断する 2.走行体（システム）は難所であった場合必要に応じてライントレースを中断する 3.走行体（システム）は難所を攻略する 4.基本系列2に進む Alt-3.コースを外れた場合 1.走行体（システム）はライン復帰を実行する 2.基本系列2に進む
例外系列	Ex-1.走行体（システム）が転倒した場合 1.走行体（システム）のフェイルセーフ機能により、走行を停止する 2.本ユースケースを終了する Ex-2.タイムアップになった場合 1.走行体（システム）は走行を停止する 2.本ユースケースを終了する
サブユースケース	
備考	



1.3攻略の優先度

アドバンストクラスでは、難所が多く全ての難所をクリアすることは難しいと感じた。さらに速いゴールタイムを出したいと考えたので1周目は出来る限り難所をスルーしたいと考えた。それらの考えをふまえ、私たちはどの難所を優先してクリアするべきか優先度を設定した。
INコースでは1周目で攻略しなければならない難所が無いため、1周目は難所をすべてスルーし、1周目のゴールから近いモーグルの優先度を1番高く設定した。直列駐車場とフィギュアLはどちらもゴールからの距離は変わらなかったが、直列駐車場の攻略のほうが比較的楽であると考えたため直角駐車の優先度を高く設定した。
OUTコースではボーナスタイムが大きく、1周目で通る必要がある難所未確定エリアの優先度を大きくし、その次に1周目のゴールから近いジャンプ台を設定した。縦列駐車場は1周目のゴールから遠く、時間内の攻略が難しいと感じたため優先度は低くなった。



1.4走行スケジュール

1周目の走行ルート　2周目の走行ルート

INとOUTでそれぞれ茶色と水色のラインを使用し1周目と2周目の走行ルートの区別をつけた。1周目は難所の攻略を出来る限り避け、速いタイムでのゴールを目指している。2周目では1.3での優先度を基に2つの難所の攻略を行う。

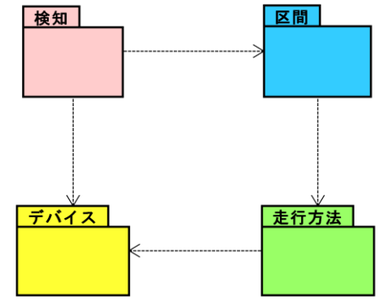
2.1設計方針

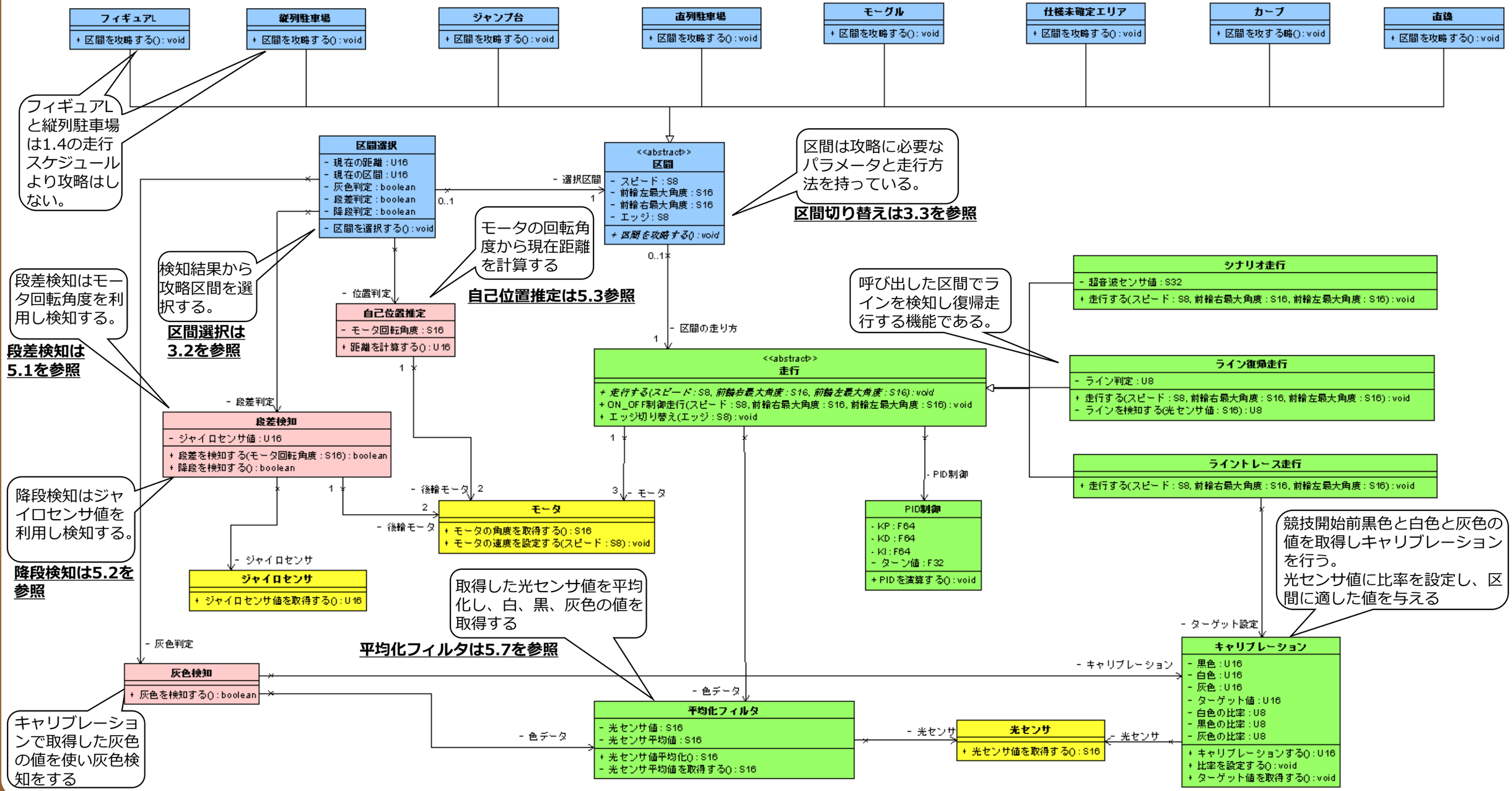
【構造の複雑さの低減】
クラスの粒度を大きくし複雑になる事を低減した。
一つのクラスが多くの操作を持つことを防ぎ、バランスの良いクラス構造を構成した。
コース図を区間ごとに分割する事により、一区間ごとの制御を可能にした。
【疎結合化設計】
クラスの役割を明確にして各クラスを独立させる事により、カプセル化を実現させた。

2.2パッケージ構造

クラスを大きく4つのパッケージに分割した。
パッケージごとに明確な役割を与えた。
パッケージの呼ばれる順番は3.1を参照。

検知	分岐条件に必要な値の検出
区間	区間ごとに攻略を分担
走行方法	主要機能と走行方法を分担
デバイス	使用機器の入出力を行う

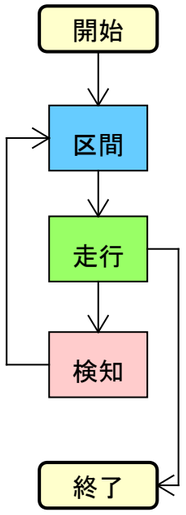




3.1全体の振る舞い

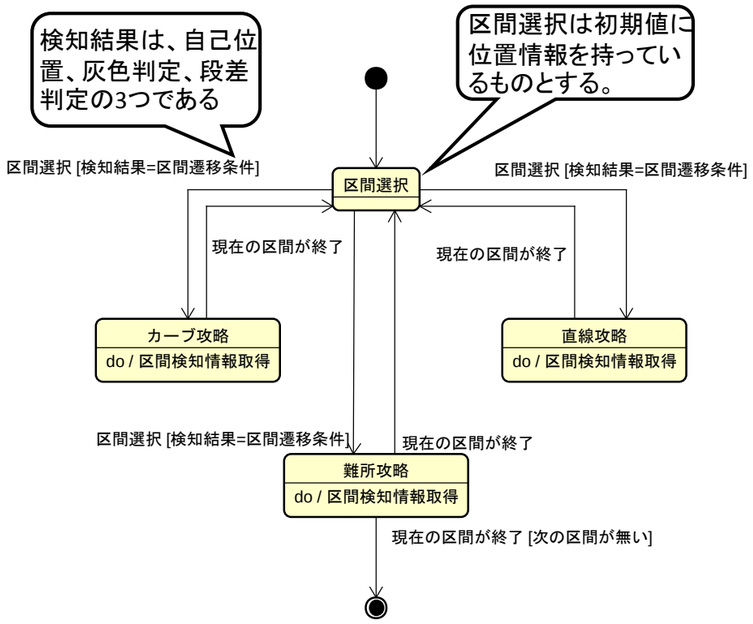
競技開始から終了までのシナリオを動的解析を用いて本モデルに記載する。全体の大まかな流れをフロー図を用いて示す。

・区間
選択した区間の持っている攻略情報から走行を呼び出す。
・走行
遷移区間がある場合は検知を呼び出す。最後の走行時に終了する。
・検知
走行しながら検知した結果を区間に送る。



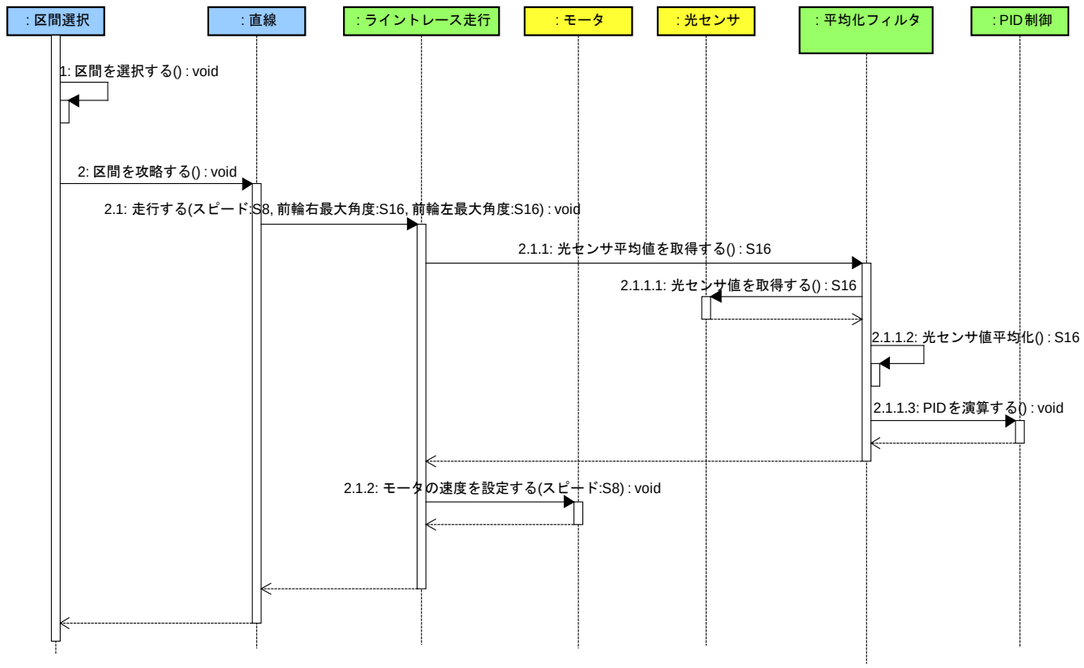
3.3区間切り替え

走行切り替えをステートマシン図に示す。
直線、難所、カーブの区間が区間選択により遷移する。



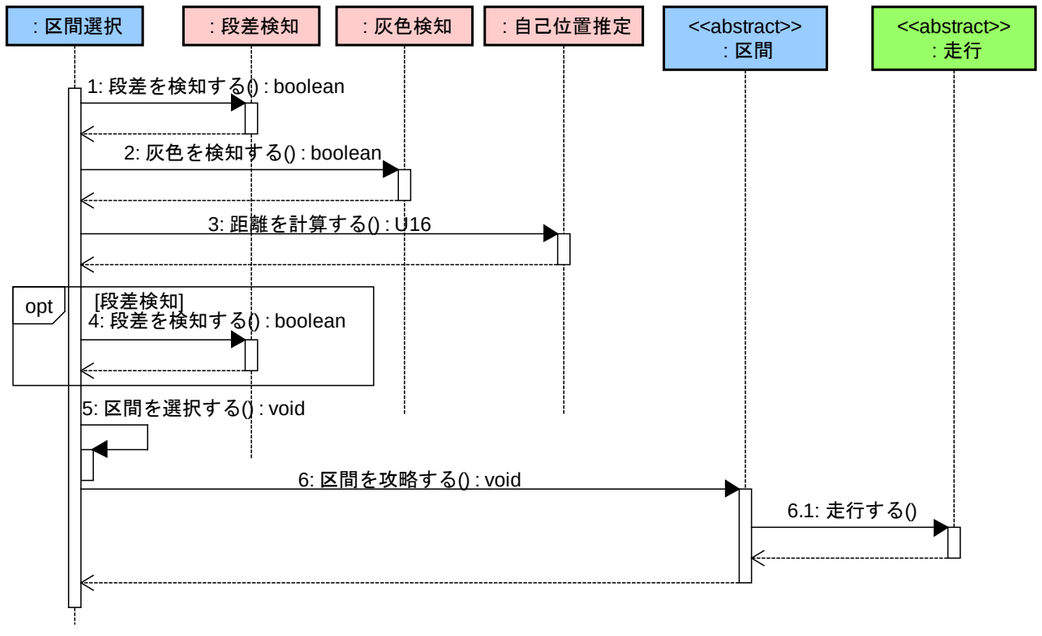
3.4基本走行

区間から走行の流れを一つの例として**区間選択⇒直線⇒ライントレース**とした。直線区間が終了後カーブへと遷移する。



3.2区間選択

区間選択と検知の関係をシーケンス図に示す。段差検知と灰色検知と自己位置推定の検知結果の値を区間選択条件に使う。**段差判定結果**or**灰色判定結果**
or**[現在の距離>=区間切り替え距離]**を条件として、条件を満たしていれば次の区間へと遷移する。



・コース図から距離を計測

【距離補正】
灰色を検知した場合、現位置情報を元から組み込まれていた位置情報に補正する。
【ソフトウェアの効率的な修正が可能】
走行区間を分別することにより、効率的に開発を行う事が出来る。

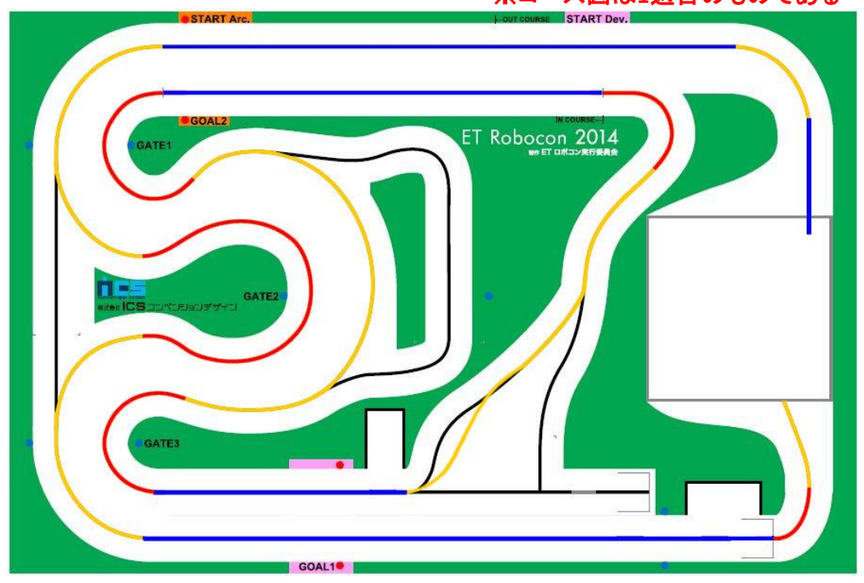
・区間説明

- カーブ
- 急カーブ
- ストレート
- 通過しない区間

各区間でパラメータを変更する事により、三輪での円滑な走行を実現することが出来る。ストレートでは**高速走行**を行い、カーブで**中速走行**を行い、急カーブでは**低速走行**を行う。

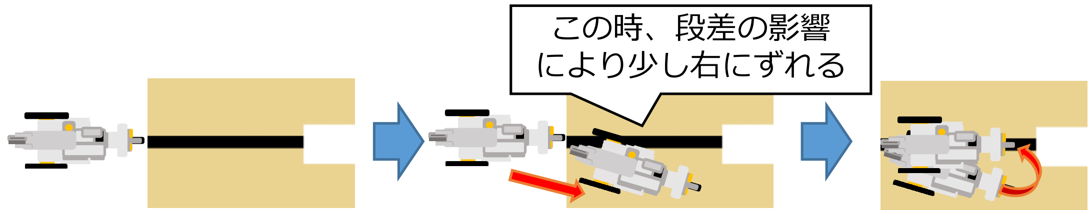
三輪走行は5.4を参照

※コース図は1週目のものである



4.1ジャンプ台攻略

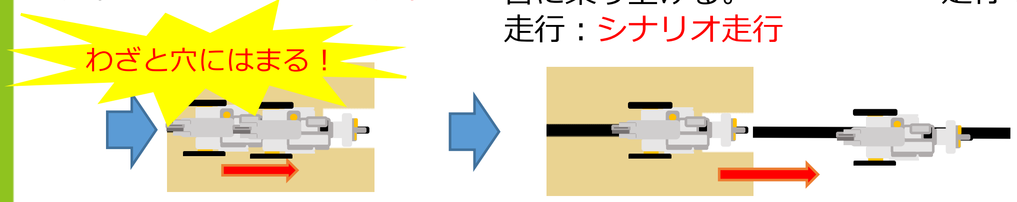
ギア比１：１ではスピードが出ないためジャンプ台を勢いよく走り抜けることが難しかった。そのため安定した攻略をするためにジャンプ台上で車体をまっすぐにし、勢いよく飛び出す必要がある。ここではその方法と手順を記述する。



段差検知により段差を検知し停止する。
走行：ライントレース走行

低速で走行し、徐々に速度を上げゆっくりとジャンプ台に乗り上げる。
走行：シナリオ走行

その場で回転しライン検知させラインに沿わせる。
走行：シナリオ走行



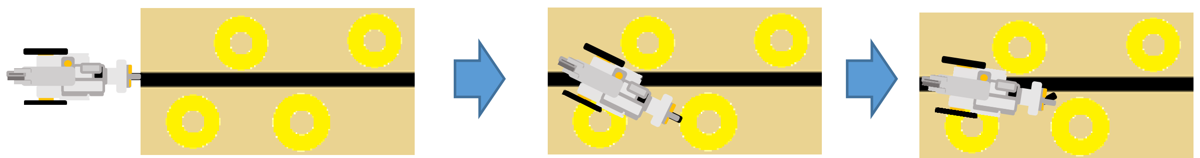
低速で走行し、前輪が前方の穴に落ちるまで前進する。
走行：ライントレース走行

後輪を高速で回転させ勢いよく飛び出し、着地後ラインへの復帰を行う。その後、次の動作に移る。
走行：シナリオ走行

穴に落ちている時は左写真のような状態となっている。
故意に穴にはまる理由：
走行体の向きと位置の誤差が矯正され、次の処理が容易になる

4.2モーグル攻略

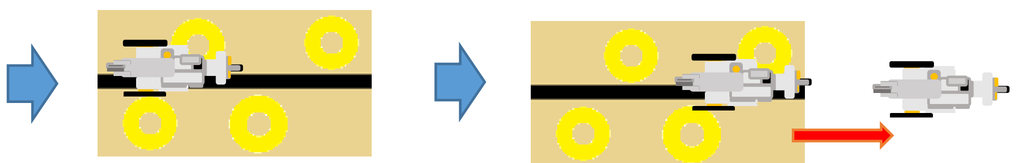
ギア比１：１の強みを活かし、中速でも力のある走行が出来ることからモーグルを安全に走行することが可能となった。ここでは、モーグルの攻略手順を記述する。



低速で走行し、段差検知により段差を検知し停止する
走行：ライントレース走行

中速で右に向かいながら段差を乗り越える
走行：シナリオ走行

中速で左に向かいながらライン検知をしてラインに復帰する
走行：シナリオ走行



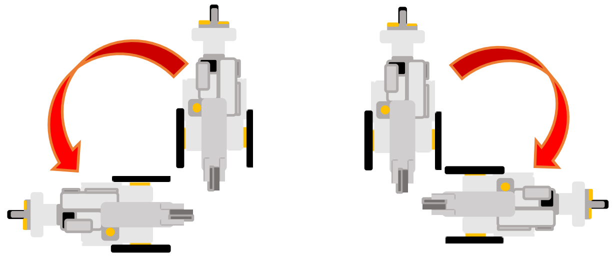
ライン検知後に左エッジでラインに沿って走行する
走行：ライントレース走行

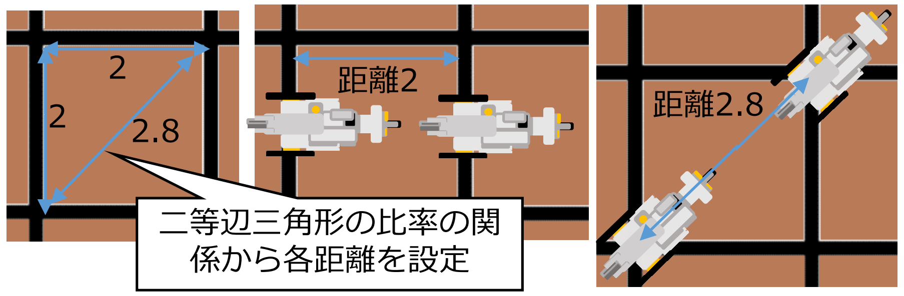
中速で走行し、段差から降りる。段差から降りたことを降段検知により確認後次の動作に移る
走行：ライントレース走行

4.3仕様未確定エリア攻略

様々な障害物配置パターンに対応するため、走行体の向き移動距離を自由に設定できる処理を関数化した。結果、仕様未確定エリアの走行ルートを容易に変更することが可能となる。

各処理の説明 :各処理の説明と動作の例を示す

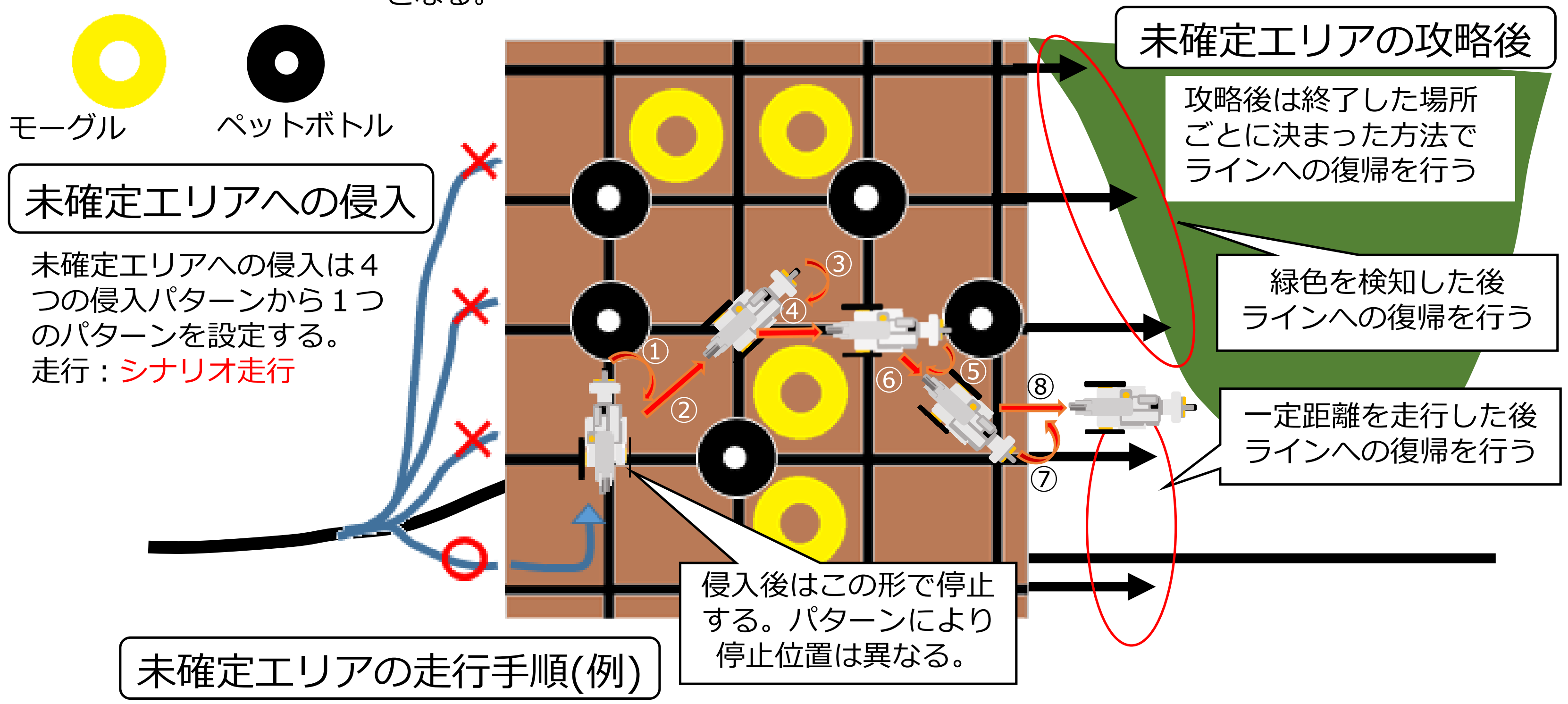




左回転 **右回転**
後輪の2つの車輪の中点を支点として任意の角度を引数として入力し回転出来る。

前進
未確定エリアの黒の線の交点と次の交点の間を２とし、任意の数字を引数として入力することでその数だけ前進出来る。

配置と走行の例： 例として以下のような配置の場合、走行体は次のような走行手順で走行することとなる。

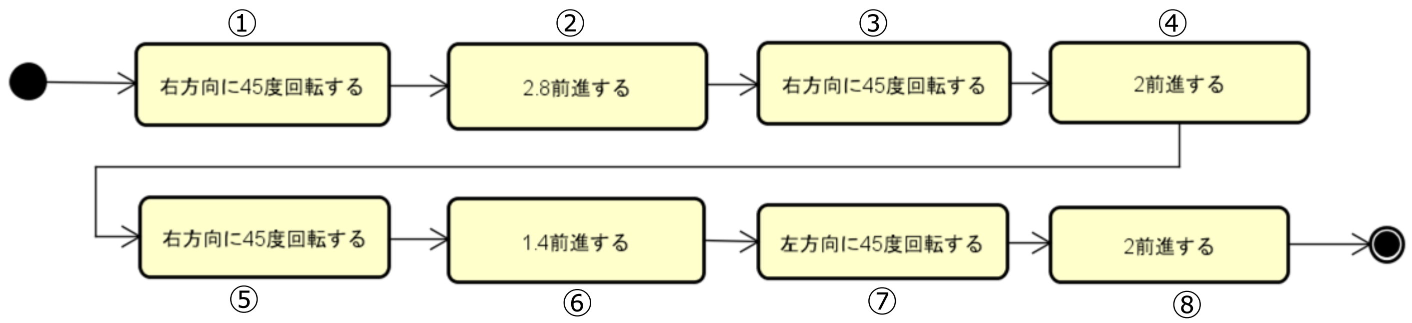


未確定エリアへの侵入

未確定エリアへの侵入は４つの侵入パターンから１つのパターンを設定する。
走行：シナリオ走行

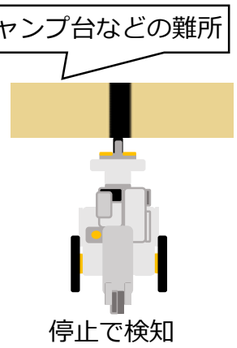
未確定エリアの走行手順(例)

未確定エリア侵入後の走行体の動作をフロー図で表現した。
各番号は上図の各番号に対応した動作を行っている。



5.1段差検知

《課題》
段差を昇る際に、どこで段差が始まるのかを正確にするため、段差を検知する必要があった。
【対策】
自己位置推定、灰色検知により、走行体が段差に近づくとスピードを低速とし、車輪を難所の段差の部分に当てる。
すると、低速走行のため車輪が停止するので、停止から2秒後に段差検知することとした。また、この技術を使用することで自己位置の補正をすることも可能となった。



停止で検知

5.2降段検知

《課題》
難所を攻略後に次の動作へ移る必要があるが、難所では進む距離が一定ではないため、距離以外による難所を攻略したという検知が必要であった。
【対策】
走行体の前輪が段差から降りたことをジャイロセンサ値を基に検知する。
結果、自己位置の補正をすることが可能となった。

5.3自己位置推定

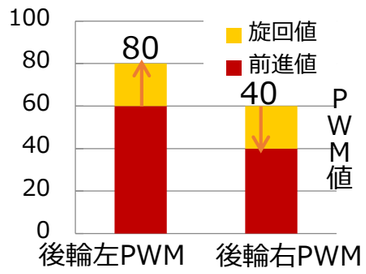
《課題》
走行体の動作変更の条件は走行距離によるものを多く使用しているが、モータの回転角度のまま条件にすると直感的に分からず扱いにくい。
【対策】
走行距離を回転角度から長さの単位に変更し、開発者にとって分かりやすいようにした。
また、自分の位置を把握することでカーブは速度を遅め、ストレートは速度を速めるといったことが出来るようにした。

走行距離(cm)
=((後輪右モータ回転角度+後輪左モータ回転角度)/2)*円周率*タイヤの半径(cm)/180

5.4三足走行（ライントレース）

《課題》
プライマリークラス用の走行体では倒立振子に渡す引数に前進値と旋回値を引数として渡すことができる。しかし、倒立振子を使用しない三足走行には、引数で旋回値と前進値を指定することが出来ない。
【対策】
旋回値と前進値を使用して三足走行でも利用できるようにした。

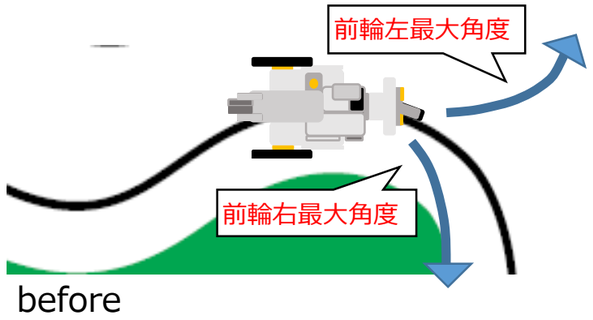
三足走行の後輪左右モータPWM値計算方法
後輪右モータ PWM値 = 前進値 −（旋回値）
後輪左モータ PWM値 = 前進値 +（旋回値）



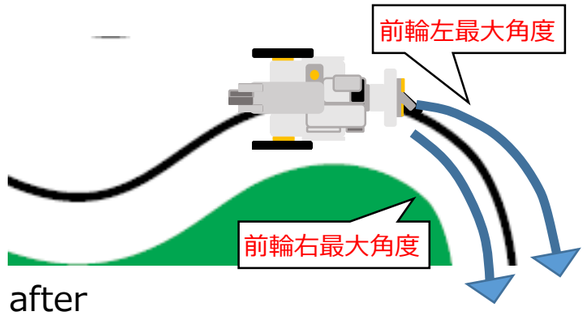
後輪左右モータのPWM値算出例
(前進値=60 旋回値=20 の場合)

5.5前輪可動域の制御

《課題》
前輪の左右の可動域が大きく設定されていると、余分な走りをしてしまいスムーズな走りが出来ない。
【対策】
区間ごとに前輪の可動域の上限を設定することで滑らかな走行をした。
走行体に対して前輪がまっすぐの場合を０度とし、
角度を右に曲がる場合には角度を正の数値、左に曲がる場合には角度を負の数値とする。



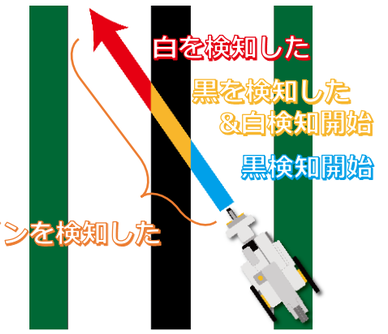
before



after

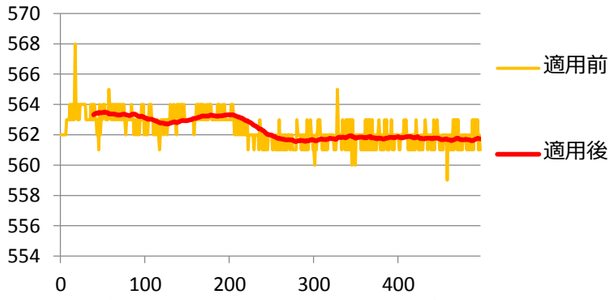
5.6ライン検知

《課題》
4.3エッジ切り替えを行う前に走行体がラインを検知する必要があった。
【対策】
走行体がコースの白色の上を走っている時に黒色のラインを検知し、再び白色を検知することで、ラインを検知することが出来る。



5.7平均化フィルタ

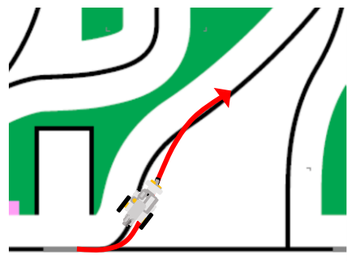
《課題》
光センサから取得した値が、光環境によっては大きな急変動が発生し正確なライントレースができない。
【対策】
40ミリ秒間の光センサ取得値を記憶し、そこから算出した平均値をもとにライントレースを行うことで、光センサ値の急な変動にも対応できるようになった。
特に水銀灯の環境下においては光センサ値の変動が激しいため、これは有効的な手段であるといえる。



水銀灯下における光センサ取得値への平均化フィルタ適用の比較

5.8シナリオ走行

《課題》
難所を攻略するため、コースを走行している途中でエッジを切り替える必要があった。しかし、ライントレースのみではエッジを切り替えることが出来なかった。
【対策】
前輪を任意の角度に制御することで、ライントレースを行うことなく予定したシナリオ通りの動作を行うことが出来る。


シナリオ走行中