1. 要求分析

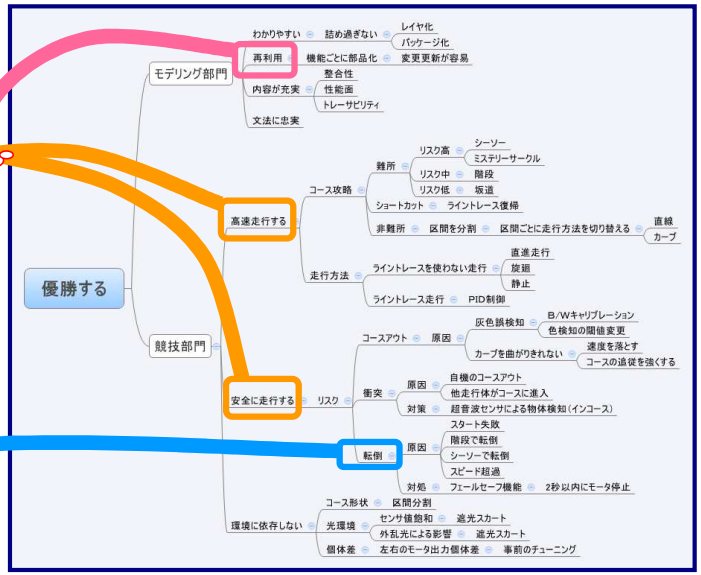
芝浦雑伎団

オージス総研

マインドマップから非機能要求、ユースケース図から機能要求を確認しました。
また、時間的な要因から差分で開発する計画をたてました。

1 目標のマインドマップ

ETロボコンにおける自分たちの目標をマインドマップで確認



テーマ **再利用しやすいモデル**

このアイコンが目印！！ 再利用しやすいモデル

パッケージ化
レイヤ化 ⇒ 2. 全体構成を参照

テーマ **安全かつ高タイムを出したい**

このアイコンが目印！！ 安全 高タイム

ショートカット実現 ⇒ 3. 走行戦略を参照
PID制御 ⇒ 5. 難所走行＆技術要素②を参照

難所攻略 ⇒ 3. 走行戦略
4. 難所走行＆技術要素①を参照

このアイコンが目印！！ 難所

テーマ **難所に対する十分な対策**

開発計画 ⇒ 開発計画

右の開発計画を参照

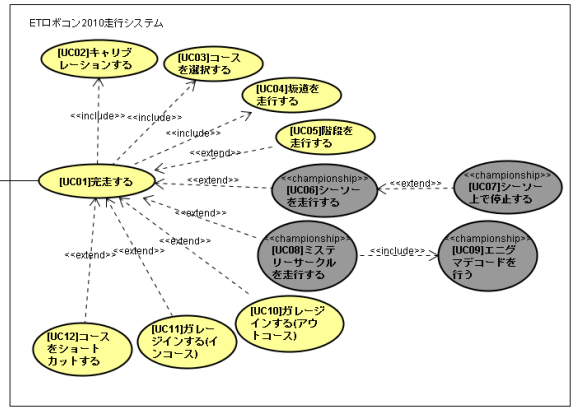
2 機能要求

ユースケース図を用いて機能要求を確認

機能実現の構造面については
2. 全体構成を参照

難所のユースケースについては
3. 走行戦略
4. 難所走行＆技術要素①を参照

ユースケース図



前提となるユースケースはinclude、選択となるユースケースはextendで表す。
また、チャンピオンシップ大会で拡張し、挑戦するユースケースには
<<championship>>のステレオタイプで表す。

ユースケース記述(一部)

基本フロー

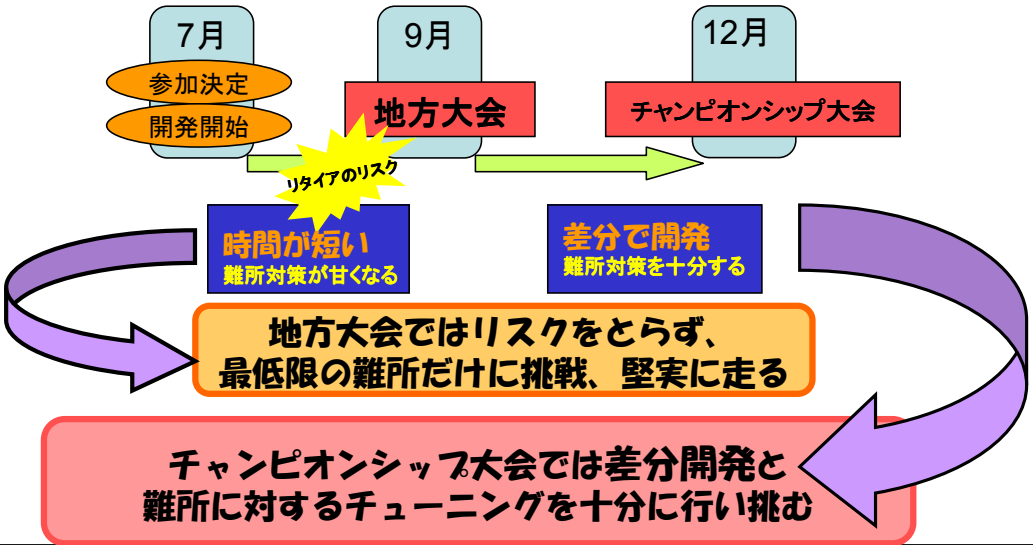
1. アクタはキャリブレーションを開始する。
2. システムはキャリブレーションする。
3. アクタはコースを選択する。
4. システムはコースに合った走行準備をする。
5. アクタはシステムに走行開始の指示をする。(※1)
6. システムは倒立制御を行い、トレース走行を開始する。
7. システムはゴールを通過する。

(※1)タッチセンサを押すことで走行開始の指示となる

3 開発計画

開発計画

例年、地方大会では難所に挑戦し、リタイアしてしまうチームが続出している。
また、我々は地方大会までの開発期間が短いため十分な難所対策が行えるとは考え難い。
そのため地方大会突破はリタイアしないことが大事だと位置づけました。



2. 全体構成

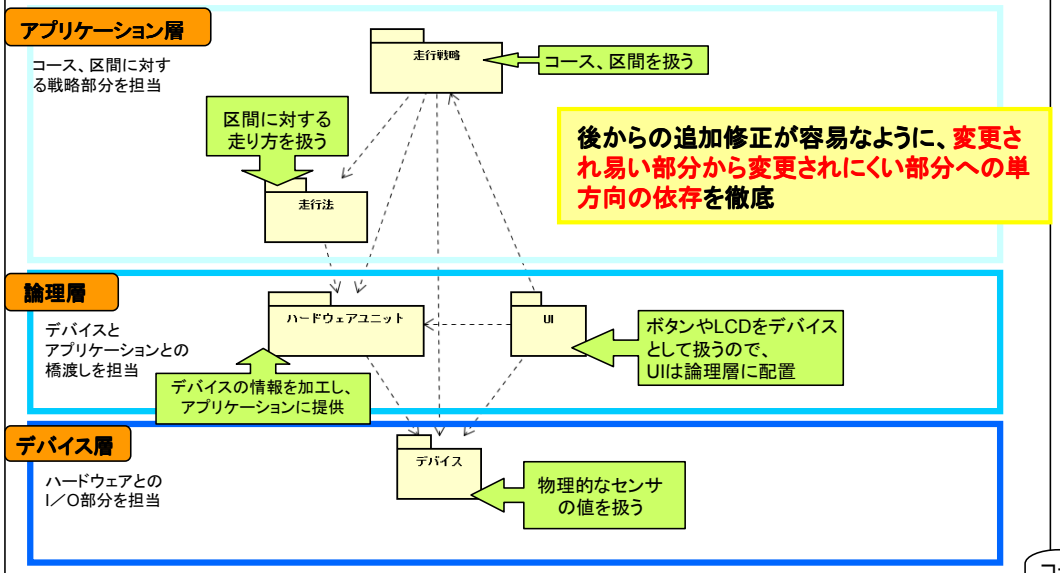
芝浦雑伎団

オージス総研

再利用しやすいモデルにするために、パッケージ化、レイヤ化を行いました。
また、アプリケーション層では後に追加開発しやすいモデルにしました。

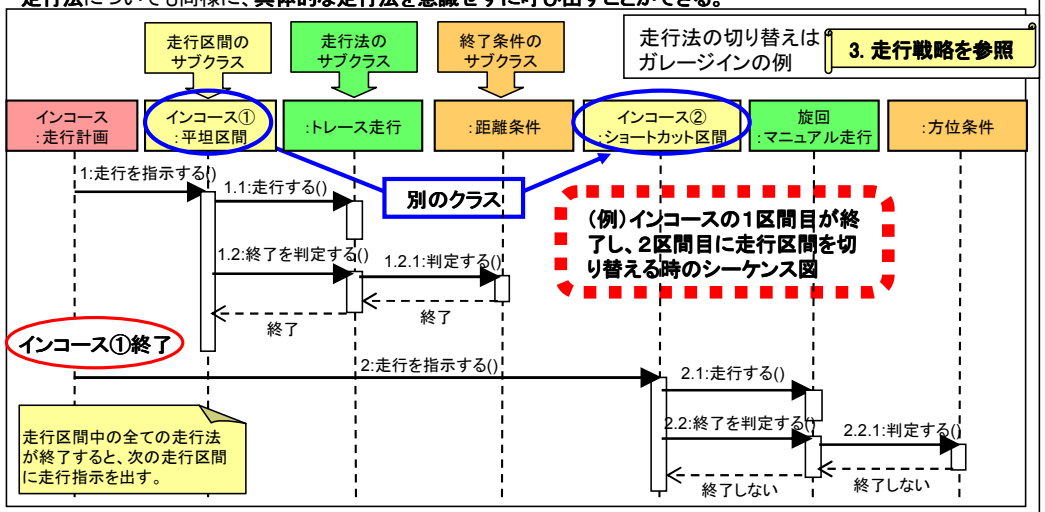
1 パッケージ構成図

開発計画　再利用しやすいモデル



3 走行区間の切り替え機構

具体的な走行はそれぞれ「**走行区間**」、「**走行法**」、「**終了条件**」を継承したサブクラスによって実現される。
例えば、**走行区間**の切り替えは走行計画が行うが、**具体的なクラスは異なっていても同じように扱うことができる。**
走行法についても同様に、**具体的な走行法を意識せずに呼び出すことができる。**



定義

「コース」・・・・・ スタートとゴールを含む、区間で構成された、走行予定経路

「走行区間」・・・ 走行予定経路となりうる区間

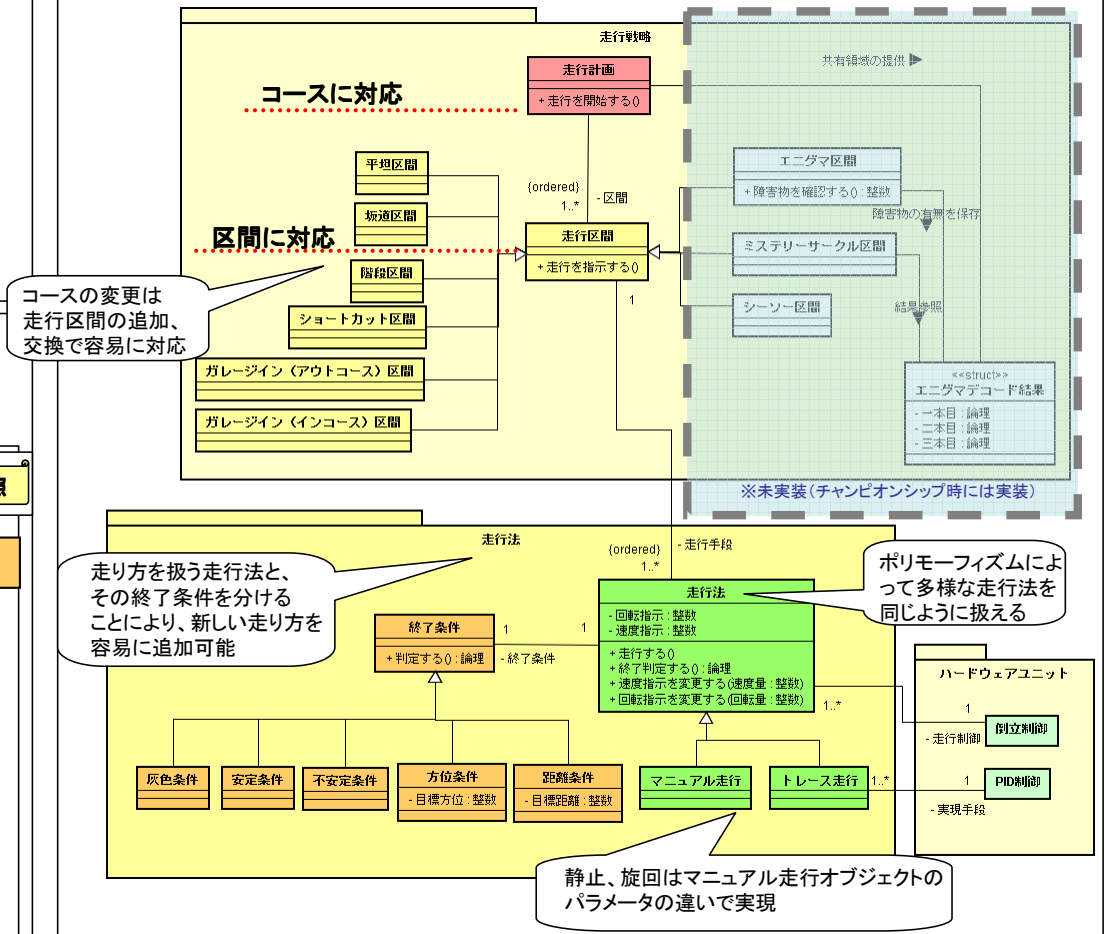
「走行計画」・・・ インコース、アウトコースを一周するための走行区間順序

区間の定義は　3. 走行戦略を参照

2 アプリケーション層の静的構造

開発計画　再利用しやすいモデル

拡張しやすい構造、修正による影響範囲が少ないパッケージ関係、クラス関係にしています。
※他のパッケージにあるクラスとの関連はこの図で表現していないものがあります。
※地方大会ではエニグマとシーソーをチャレンジしないので実装はしませんが、モデル上では考慮しました



3. 走行戦略

芝浦雑伎団

オージス総研

コースを分析した結果、安全と高タイムの両立を目指すべく選択した戦略を紹介します。
詳しい難所の攻略法は、次の「4. 難所走行＆技術要素①」にあります。

1 安全かつ高タイムを出すための戦略は？

高タイムで走行するには、どうすればよいのだろう？

1. ハイレベルな難所をクリアし、ボーナスタイムを得る。
2. より短い時間で一周する。

地方大会では**絶対にリタイアしたくない**ので、
2.のアプローチを重要視。
より短い時間で走破するために、ショートカットに挑戦する。

2 区間分けの基準

1. 階段・シーソー・ミステリーサークルやショートカットなど、戦略的にルートを選択できる部分は1区間とする。
2. 坂道やガレージインなど、特殊な制御が必要な部分は1区間とする。
3. 1と2以外の部分で、連続した部分をそれぞれ1区間とする。

3 挑戦する難所の選定

難所　安全　高タイム

リタイアの要因は極力排除したい。しかしタイムを縮めるには難所も必要。 ⇒リスクの低い難所を選択する

ボーナス対象	ボーナス	リスク	難易度	備考	採用？
シーソー通過	20秒	転倒 高	難	アウトコース競技者のみ	×
シーソー上停止	60秒	転倒 激高	激難	アウトコース競技者のみ	×
階段通過	20秒	転倒 中	易	アウトコース競技者のみ	○
ミステリーサークル	40秒	誤認識時 高	中	インコース競技者のみ	×
衝立部分のショートカット	0秒	衝突リスク有	中		○
ガレージ・イン（アウトコース）	20秒	なし	中	ペナルティなし	○
ガレージ・イン（インコース）	20秒	なし	中	ペナルティなし	○

競技規約　5.4.6版対応

選定の理由
階段は転倒のリスクがあるが、シーソーに比べれば低く、攻略法も立てやすい。
ガレージインはゴール後のため、リタイアのリスクはない。
ショートカットは衝突の場合責任が生じるが、相手コースに入らなければリスクは低い。

4 ショートカット

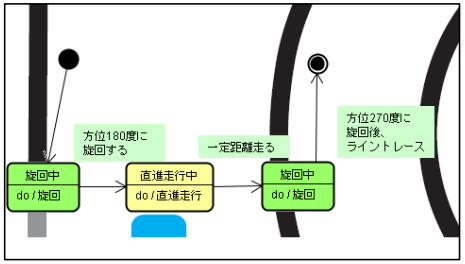
高タイム

タイムを稼ぐ方法としてショートカットを選択したが、ショートカットの方法にもさまざまな方法が考えられる。
前進しながらカーブする方法もあるが、今回は一旦停止して旋回する方法を採用した。

採用したショートカット方法

1. ショートカット開始地点で旋回し、走行体をショートカット先の方向に向ける。
2. ショートカット先の地点まで、一定距離を直進走行する。
3. 復帰したいラインの方向に旋回する。

旋回・直進については
5. 難所走行＆技術要素②を参照



メリット
- 直感的にわかりやすい方法
- 方向と距離を指定するだけでよく、プログラムが組みやすい。

デメリット
- その場で旋回すると、走りながら曲がるよりブレが大きい。

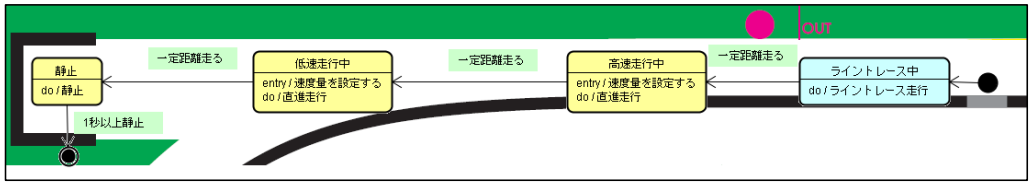
⇒デメリットはあるが、シンプルな方法を採用した。

5 ガレージイン

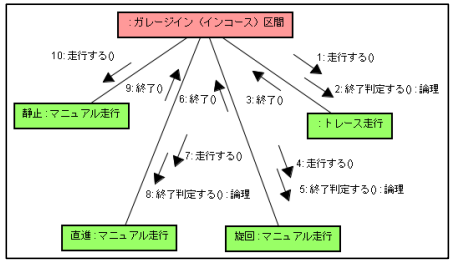
難所　高タイム

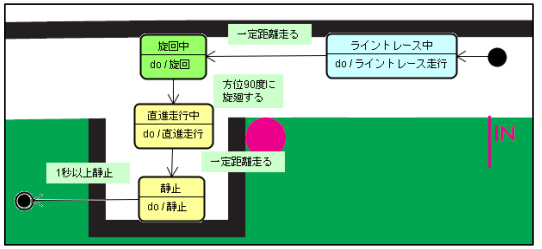
旋回・直進については
5. 難所走行＆技術要素②を参照

アウトコースの場合（下図）
ガレージがゴール後の直線の延長線上にあるため、直線部分でライントレースし、その方向に直進する。
10秒間という制限があるため、最初は高速で走行し、一定距離から減速し、ガレージ内で静止する。



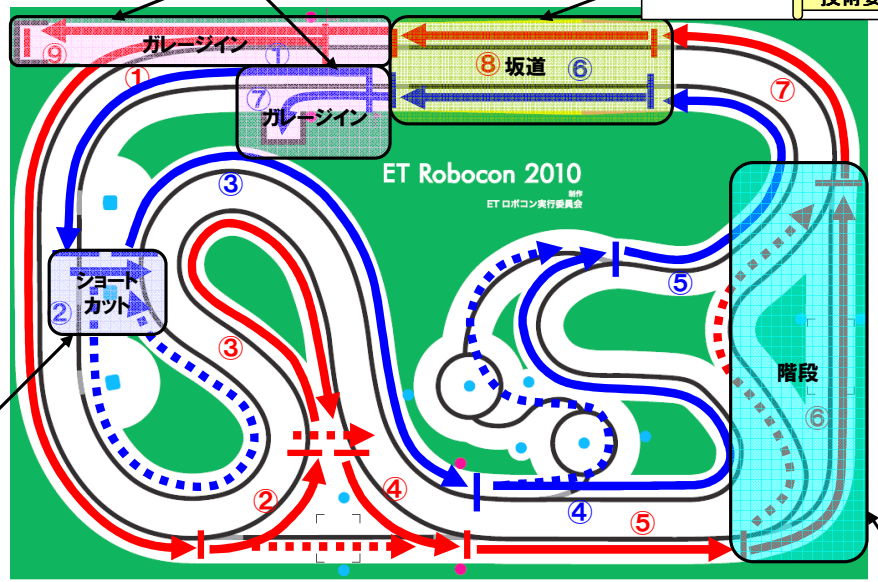
インコースの場合（右図）
ガレージがゴール後の直線から左に90度曲がったc地点にあるので、一定距離を進んだ後に旋回、直進して静止させる。





実現のメカニズム（左図）
ここでは**走行区間**が複数の**走行法**を持ち、一つの走行法が終了すると次の走行法を実行する。**制御は走行法オブジェクトが行い、ポリモーフィズムによって動作が変更される。**

坂道 → 4. 難所走行＆技術要素①を参照



凡例　アウトコースの区間　インコースの区間　代替区間

階段 → 4. 難所走行＆技術要素①を参照

4. 難所走行＆技術要素①

芝浦雑伎団

オージス総研

難所のうち、必ず通らなければならない坂道と、今回挑戦することにした階段の攻略法について紹介します。搭載が推奨されているフェールセーフ機能の紹介もこのシートで行います。

1 階段

難所　安全　高タイム

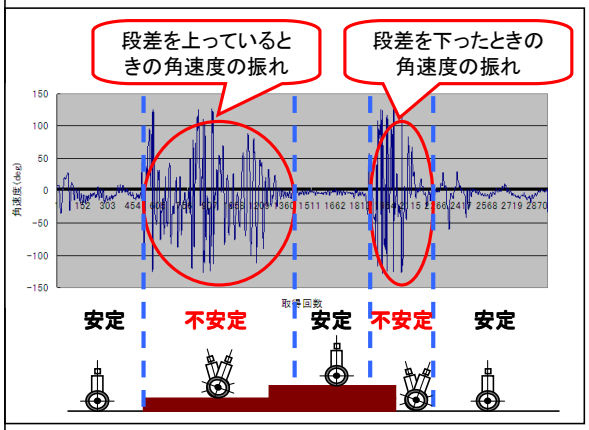
第一回試走会にてテストプログラムを走行させてみた。その結果様々なことが分かった！
走行を見た結果の問題点は以下のようになる。

問題点

段差を越えるとき
- 段差に対して斜めに進入する ⇒ **転倒 / コースアウトの危険性！** → 階段進入時の問題
- 段差に衝突し、バランスを崩したまま速度を上げる ⇒ **転倒の危険性！**

段差を下りるとき
- 段差を下りた瞬間にバランスを崩し、そのまま速度を上げる ⇒ **転倒の危険性！** → 階段の上り下りの問題

また、テストプログラムで運良く成功したときの角速度のログから様々なことが分かった。



段差の上り、下り時は角速度の振れが大きくなる

不安定になったら速度を落としてバランスを取るようにすればよいのでは！

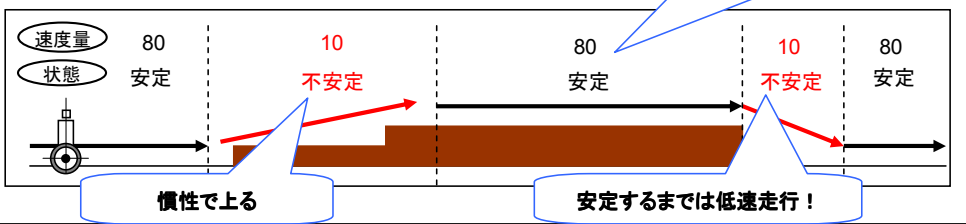
上記の結果から、攻略法が決定！

攻略法

- 段差に対して斜めに進入 → ライントレース（PID制御）を使用する！ → 階段進入時の問題 に対応
- バランスを崩したまま速度を上げる → 5. 難所走行＆技術要素②の「PID制御」を参照
- 角速度から不安定状態を検知！
- 不安定状態では速度を落として走行！

→ 階段の上り下りの問題 に対応

図にしてみた！



2 坂道

難所　安全

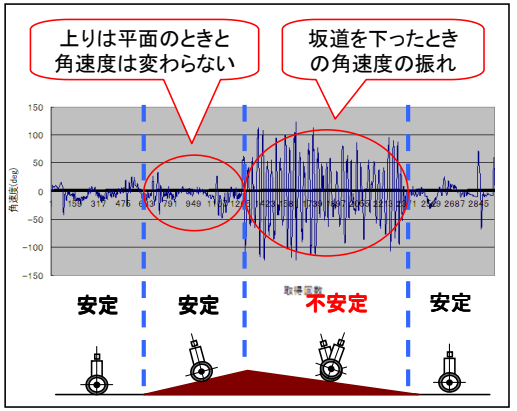
第一回試走会でライントレース（PID制御）を用いて走行させてみた。その結果、様々なことが分かった！
走行を見た結果の問題点は以下のようになる。

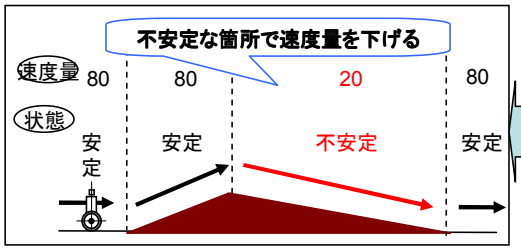
問題点

坂道区間中
- 坂道進入時の灰色領域で機体が迷走 ⇒ **コースアウトの危険性！** → 坂道進入時の問題

頂上から下り区間の間
- 斜面の切り替わりでバランスを崩したまま、上りと同じ速度量で前進させる ⇒ **転倒の危険性！** → 坂道の下りの問題

成功したパターンは多数存在したが、走りを見ていると安全とは言えない･･･。
成功したときの角速度のログを解析してみた。





図にしてみた！

斜面の下りで角速度の振れが大きくなる

坂道の下り区間になったら速度を落としてバランスを取ればよいのでは！

上記の結果から、攻略法が決定！

5. 難所走行＆技術要素②の「灰色対策」を参照

- 坂道に対して斜めに進入 → 灰色対策を行なう！ → 坂道進入時の問題 に対応
- バランスを崩したまま速度を上げる → 角速度から不安定状態を検知！ → 坂道の下りの問題 に対応
- 不安定状態では速度を落として走行

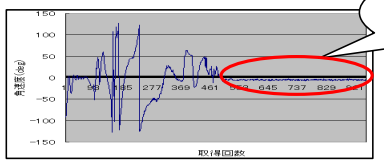
3 フェールセーフ

安全

転倒時の対策

1. 角速度から転倒を検知する。
2. 転倒を検知した場合、モータに走行指示を与えても回転しないように停止フラグを切り替える。

転倒時の角速度

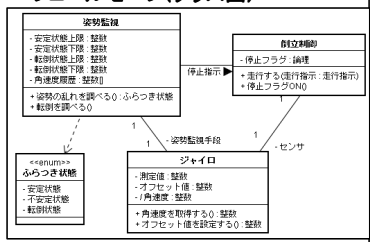


転倒するとー定の幅の角速度しか測定されない

この状態を検知したときモータの回転を停止する！

これらの振る舞いでフェールセーフ機能を実現！

フェールセーフ（クラス図）



5. 難所走行＆技術要素②

芝浦雑伎団

オージス総研

今回当チームが採用している技術要素についての解説を行います。

■PID制御

安全　高タイム

PID制御とは！

フィードバック制御の一種であり、制御量を出力値と目標値との僅差、その積分、および微分の三つの要素によって行う方法

PID制御の「P」「I」「D」とは！

P(Proportional) ･･･比例成分 → 目標値に近づける！
I(Integral)･･･積分成分 → 比例成分の残留誤差を修正！
D(Derivative)･･･微分成分 → 急激な変化が起こったときの修正！

PIDの公式！

$\Delta x(t) = Kp * P(t) + Ki * I(t) + Kd * D(t)$
$\Delta x(t)$: 制御量 Kp : 比例ゲイン Ki : 積分ゲイン Kd : 微分ゲイン

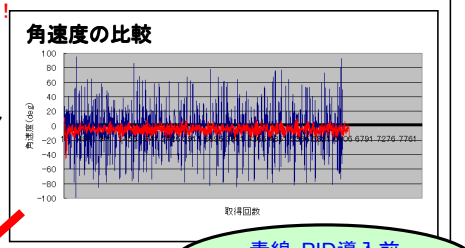
各成分の式

$P(t)$ = 目標値 – 出力値 = T0 – O(t)
$I(t)$ = 偏差の累積値 = $\Sigma P(x)$
$D(t)$ = 一定時間の変化量 = $P(t) – P(t-1)$

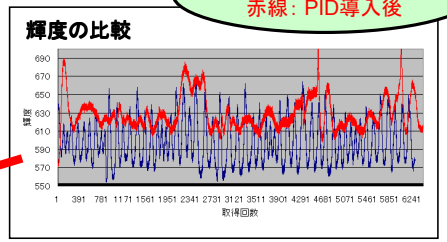
サンプルコードとPID制御を導入したコードの比較

角速度
導入前より導入後のほうが振れ幅が小さい
⇒ PID導入で安定性が向上

輝度
導入前より導入後のほうが振れ幅が小さい
⇒ PID導入でラインに対する追従性が向上



青線:PID導入前
赤線: PID導入後



■灰色対策

安全

問題①

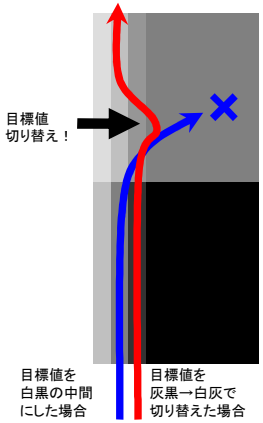
線が灰色になっている箇所で、**黒と白の中間値**をPID制御の目標値にしていると迷走してしまう。

原因

黒と白の中間値は**灰色の値**に近いため、灰色を境界線と誤認識してしまう。

解決

普段は**黒と灰色の中間値**を目標にし、**黒と灰色の中間値**が検出されなくなったら、**灰色と白の中間値**を目標値にする。



問題②

灰色マーカーで分岐している箇所で、進みたい方向に進ませることができない。

原因

黒と灰色の中間値を目標値にしていると、分岐では必ず黒線の側に向かうことになる。

解決

灰色側に進みたい区間では、あらかじめ目標値を**灰色と白の中間値**にしておく。

■直進走行

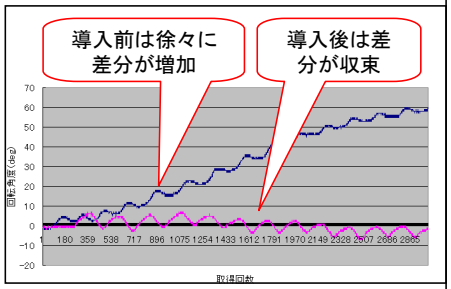
安全

左右のモータ個体差

直進する指示を与えても、機体が直進しない！
→ 指示が間違っていないなら、H/Wに問題がある！
実際に左右のモータの回転数の差分を調べてみた！

左右のモータ回転数のズレが判明
⇒ **同じ回転数に近づけるために、モータ出力に係数をかける！**

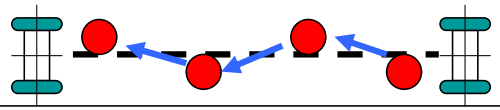
係数導入結果はこちら！
（グラフは左右モータ回転角度の差分）



方位修正

問題点　コースの路面の状態によって、直線指示を出しても直進しない場合がある。

解決法　左右のモータの回転数から機体の向いている方位を算出し、目標の方位に向かうように回転補正を与えて軌道修正！



■車体の走行距離・角度算出

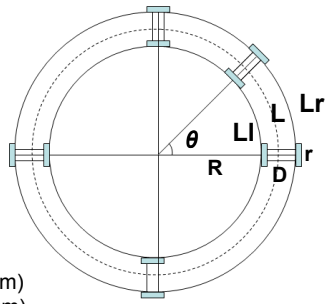
モータエンコーダの値を使用して、
機体の向いている角度と走行距離を算出する。

走行距離

左右のモータの走行距離は
$Ll = 2\pi * 半径 * 車輪回転数 = 2\pi * r * el / 360$
$Lr = 2\pi * 半径 * 車輪回転数 = 2\pi * r * er / 360$
機体の走行距離は両輪の走行距離の輪の半分とする
$L = (Ll + Lr) / 2$ [cm]

車体回転角度

$\theta = (Lr – Ll) * r / D$ [deg]

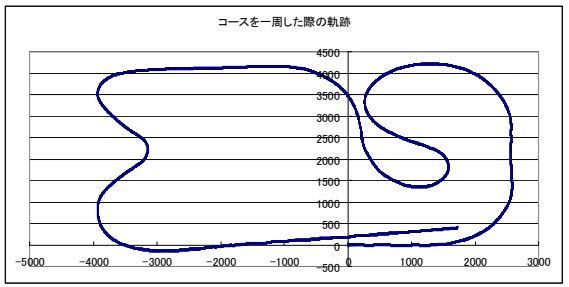


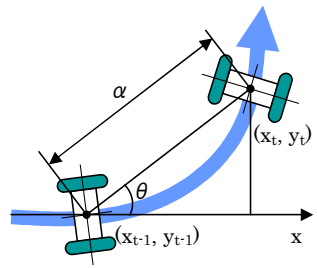
θ : 車体回転角度(deg)
R : 内円半径(cm)
r : 車輪半径(cm)
D : 車輪間距離(cm)
Ll : 左モータ走行距離(cm)
Lr : 右モータ走行距離(cm)
el : 左モータ回転角度(deg)
er : 右モータ回転角度(deg)

■ログの可視化

ログを有効活用するには、どこで変化が起こったかを知ることが重要

コースのどこを走っているときのログかを可視化したい！





走行ログから座標(x_t, y_t)を求めて
グラフにプロット

$x_t = \alpha \cos\theta + x_{t-1}$
$y_t = \alpha \sin\theta + y_{t-1}$