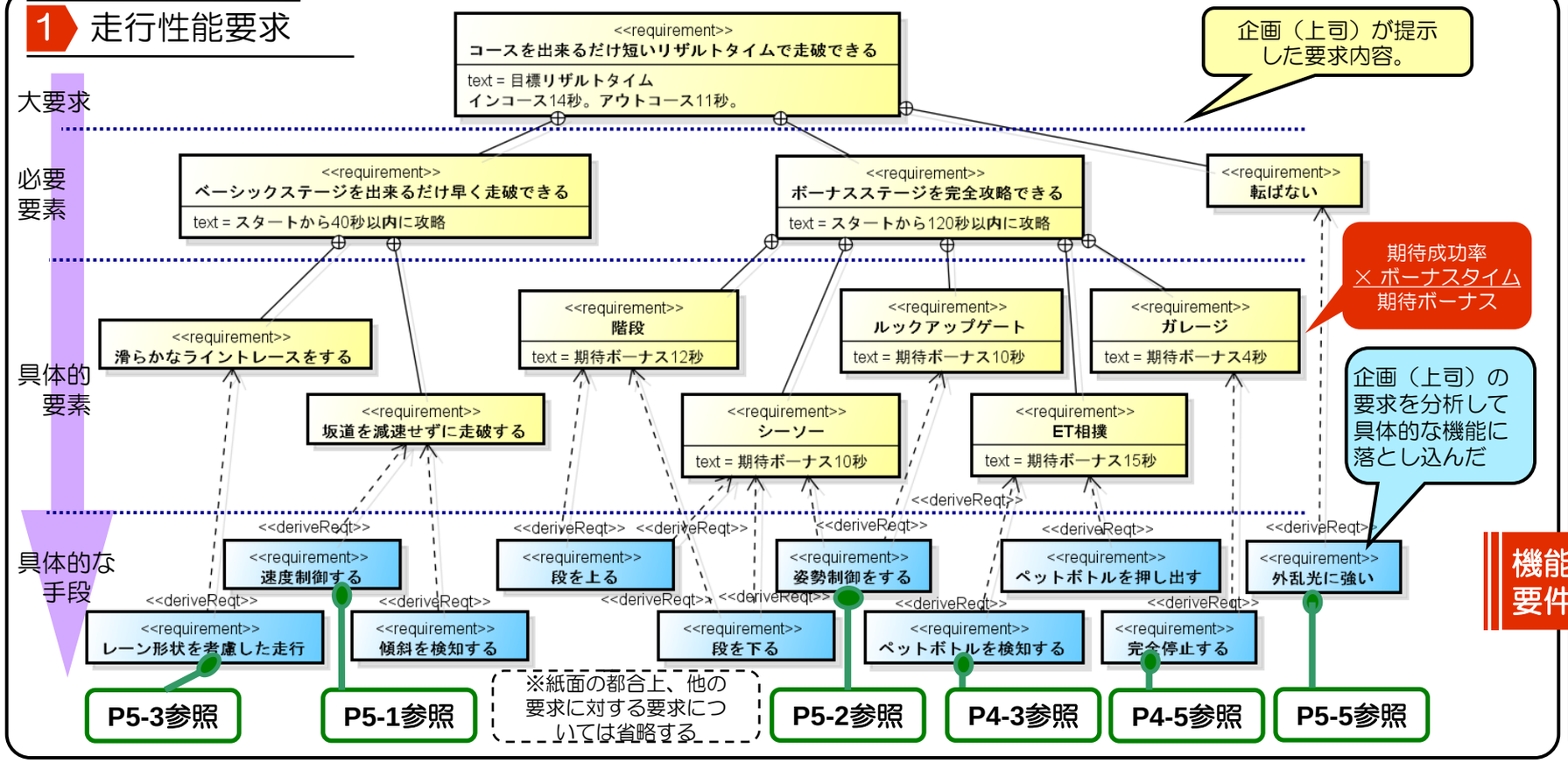
1 要求分析

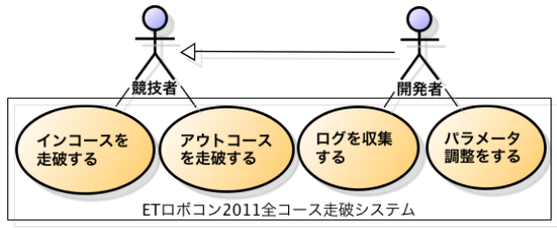
要求を分析してユースケースを決定。ソフトウェア品質向上のための支援ツールを用意する。

HAMA ROBO 2011

1 走行性能要求



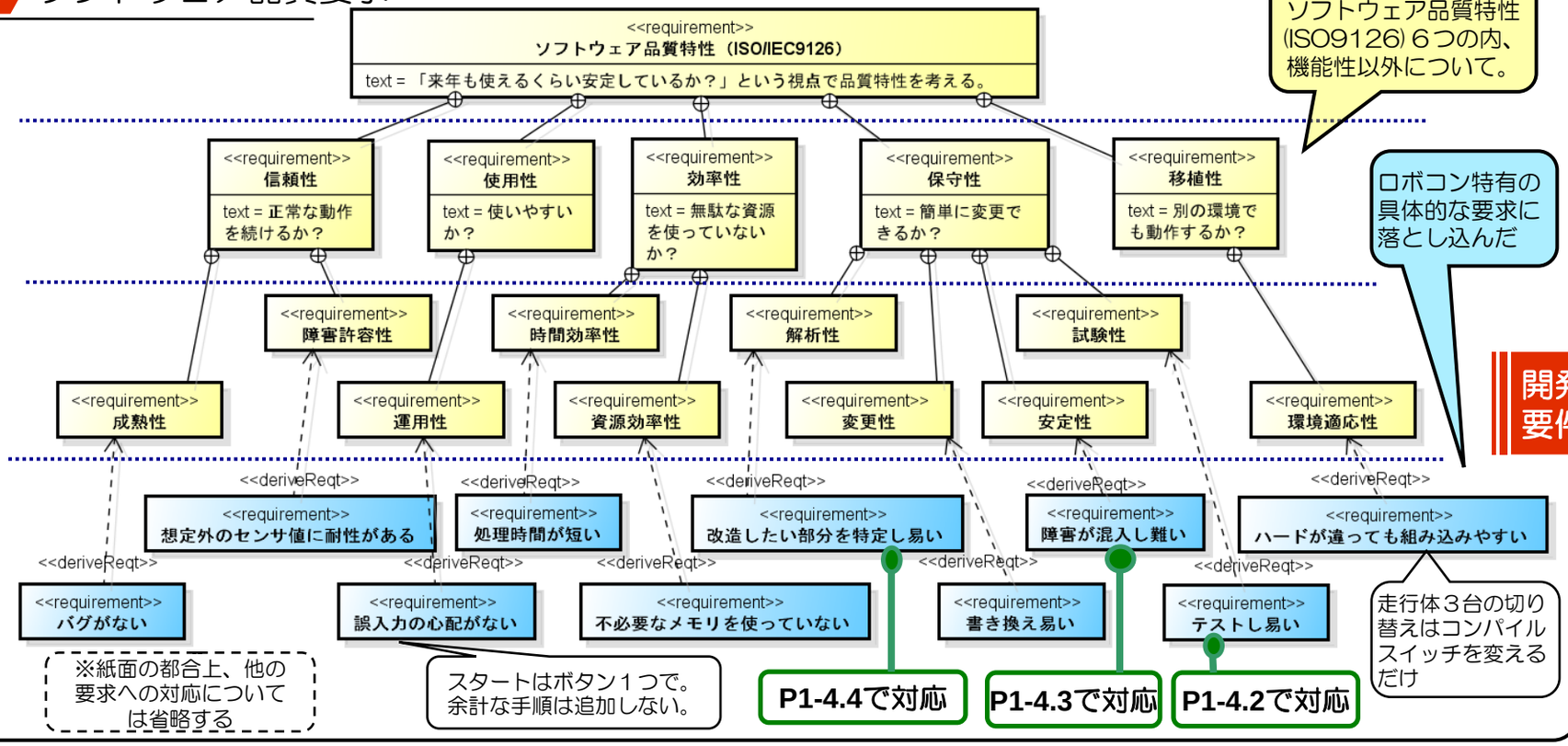
2 ユースケース



ユースケース	アウトコースを走破する。
概要	システムがアウトコースを走破する。
アクター	競技者
事前条件	走行体がスタートライン上で停止している。
事後条件	走行体がゴール地点で停止している。
基本系列	1. 競技者は走行体のタッチセンサを押す。 2. システムはライントレースをする。 3. システムはルックアップゲートを攻略する。 4. システムはET相撲を攻略する。 5. システムはガレージインをする。
例外系列	転倒した場合、モータを止める。

※紙面の都合上、他のユースケース記述は省略する。

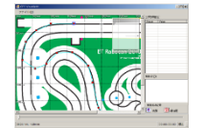
3 ソフトウェア品質要求



4 開発支援

4.1 GPS Visualizer ～走行軌跡可視化ツール～

GPSが出力した座標の時系列データを読み込むことで走行軌跡を描画する「可視化ツール」を独自に開発。



4.2 Google Test & Mock ～テストフレームワーク～

Google TestとGoogle Mockを使って、実機がなくても単体テストと結合テストを行えるようにしている。また、Jenkinsを使って継続的インテグレーションを行い、テストを自動化している。

4.3 Git ～バージョン管理ツール～

去年はSubversionを使っていたが、今年は分散型バージョン管理ツールであるGitを使ってバージョン管理をしている。バグが混入する前に戻したり、調査に使えている。

4.4 Doxygen ～ドキュメンテーションツール～

ソースコードに Doxygen コメントを記載し、HTML形式のドキュメントを自動生成できるようにしている。コード構成の理解に役立つ。

2 構造

変更容易性を追求した設計。

1 設計方針

システム機能要求で洗い出した要素を実現するため以下の方針で設計をおこなう。

①ドメイン分割をおこない、ドメインごとに**パッケージ化**をおこなう。
②抽象クラスを用いた“走法”と“終了条件”の組み合わせによる**基本シーケンス**により、すべての難所攻略を可能にする。

高い解析性の実現
高い変更性の実現

基本シーケンスの詳細についてはP3を参照

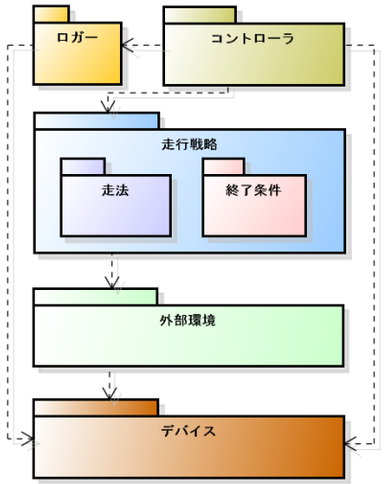
分析

分析

各パッケージの責務

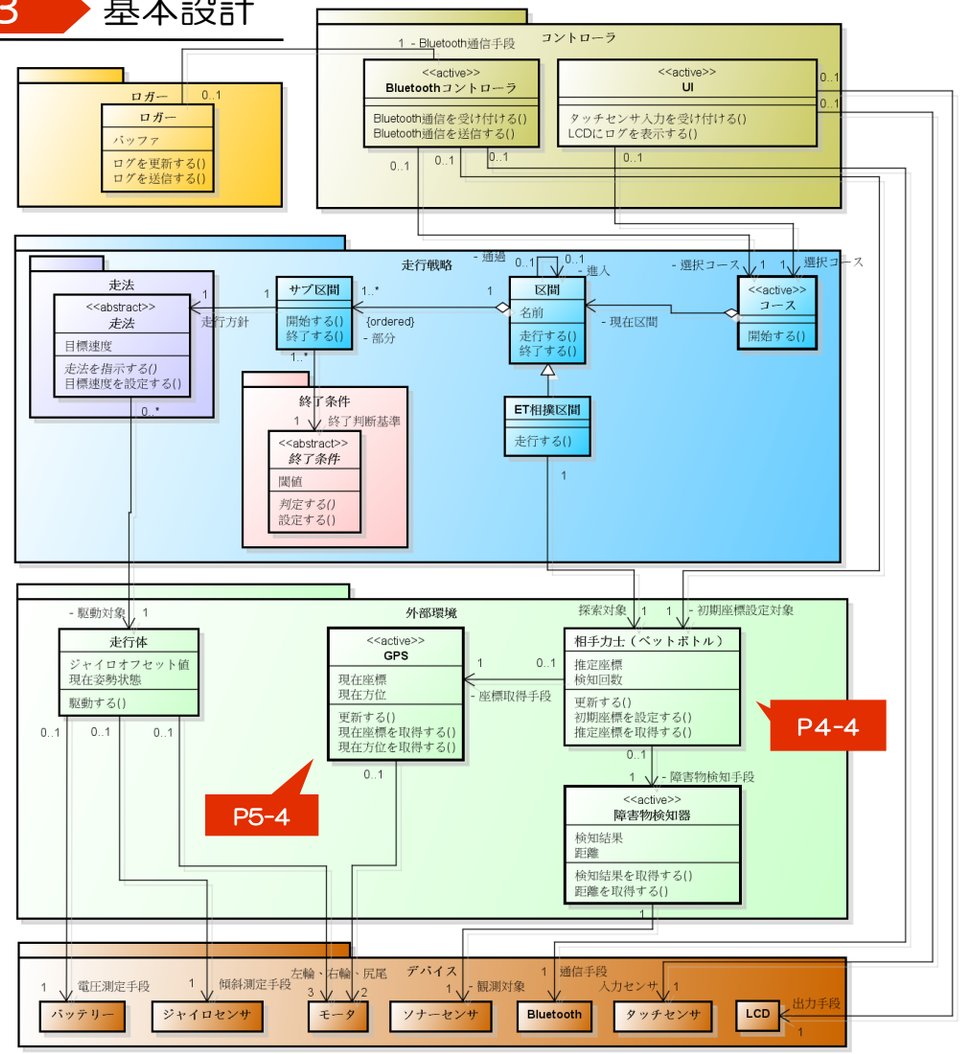
パッケージ		責務
コントローラ		ユーザとのデータの送受信を管理する。
ロガー		開発・評価用のログの管理をする。
走行戦略		区間・サブ区間にあった走法と終了条件を管理する。
	走法	外部環境に合わせて適切なベクトルを計算する。
	終了条件	サブ区間終了条件を判断する。
外部環境		デバイスから得た情報を整理・管理する。
デバイス		各センサ類を管理する。

パッケージ図



詳細

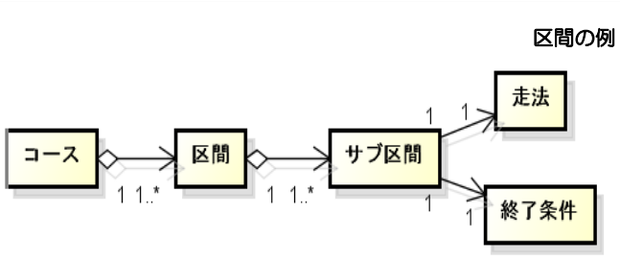
3 基本設計



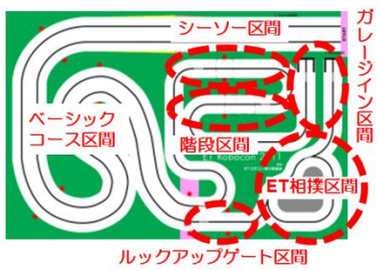
※　ロガーの関連は、紙面の都合上省略しています。

2 概念モデル

コースは特徴的な区間で構成され、その区間は”走法”と”終了条件”の組み合わせの連続で攻略される。

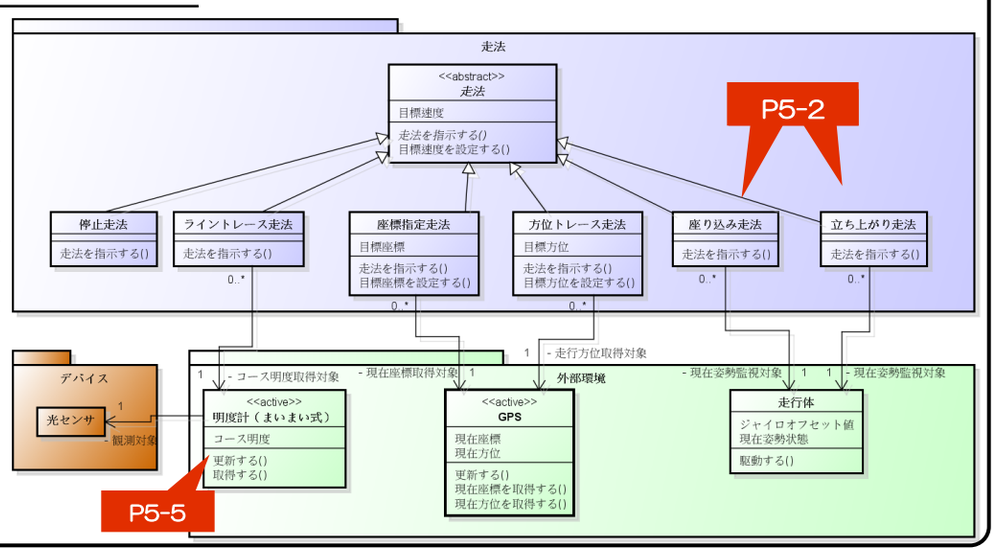


区間の例



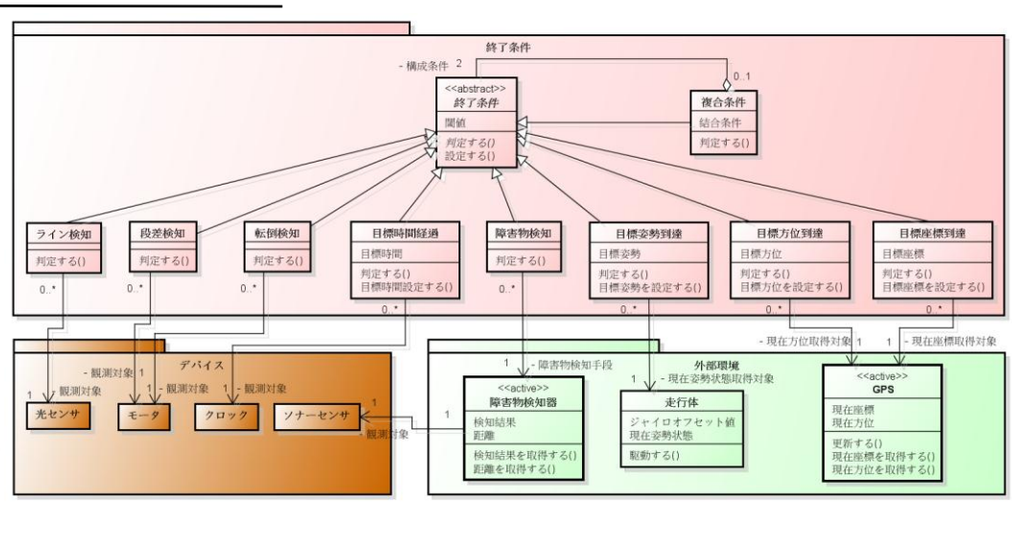
4 走法パッケージ

走行体の、駆動方法を決定するパッケージ。
駆動方法の決定アルゴリズム（ライントレース、方位トレースなど）を、クラスとして部品化することで、これらの部品の組み合わせにより、様々な走りを実現できる。
さらに、インターフェイスの統一で拡張性も確保。



5 終了条件パッケージ

走行体の状態（現在座標、現在方位など）と、外部環境（障害物の検知など）から、区間の区切り（走法の切り替えタイミング）を判定するパッケージ。
複合条件(Compositeパターン)の導入により、複雑な条件にも柔軟に対応可能。



3 振る舞い

並列して処理を行う４つのタスクの振る舞いを記す。

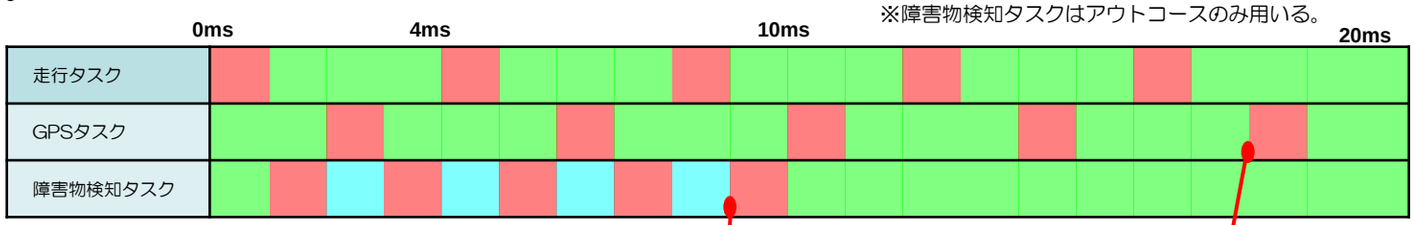
1 タスク周期

タスク実行時間・遅延の検証をもとに以下の４タスクを平行処理する。

高 優先度 低

タスク名	周期(ms)	タスク実行時間(ms)	考察
1. 走行タスク	4	0.2 ～ 1.1	バランス制御APIの制約により4ms毎に倒立制御をおこなう。用いる走法により大きくタスク実行時間が異なる。
2. GPSタスク	4	0.9 ～ 1.1	現在座標、及び現在方位を更新する。精度を高めるため独立したタスクとした。
3. まいまいタスク	10	～0.1	LEDの明滅を繰り返し、外乱光に強いセンシングをおこなう。“消灯→点灯”に8ms程度かかるというハード上の制約があり、走行タスクと周期が異なるため独立したタスクとした。安全のため10ms周期で回している。
4. 障害物検知タスク	20	4.9 ～ 5.1	相手力士（ペットボトル）を検知し、推定座標等を更新する。ソナーセンサを利用するため実行時間が長く、他の処理への影響を考慮し、優先度の低い独立したタスクとした。
他タスクに結合 5. UI受付タスク	10	～0.1	ユーザからの操作を受け付ける。OSのオーバヘッドを考慮し走行タスク内で実行することとした。
他タスクに結合 6. Bluetooth通信受付タスク	10	～0.1	Bluetooth通信の送受信を受け付ける。OSのオーバヘッドを考慮し走行タスク内で実行することとした。

※障害物検知タスクはアウトコースのみ用いる。



タイミング図：実行タスクの遷移を実録データをもとに記載

RUNNING	タスク実行中
WAITING	待機中
READY	準備中

他タスクに割り込まれている

※まいまいタスクは実行時間が短いため割愛

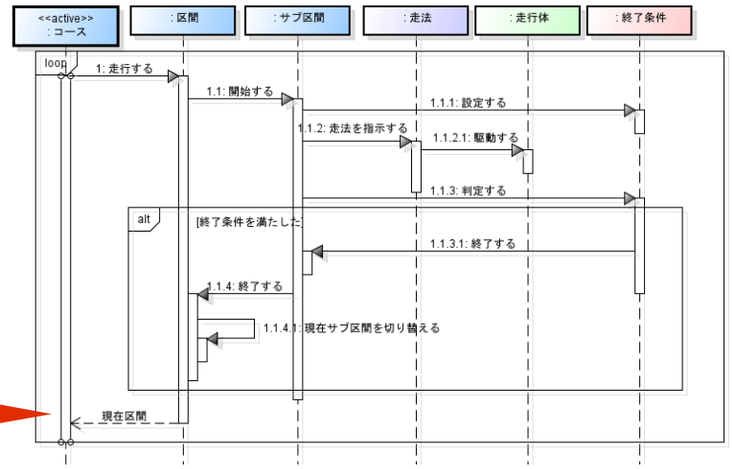
GPSタスクのアラームを走行タスクと2msずらす事により、互いのタスクへの干渉を無くす

※実際はWAITING（緑）状態からRUNNING（赤）状態への遷移の間にREADY状態（青）があるが十分短いため図から省略する

2 走行タスク

2.1 基本シーケンス

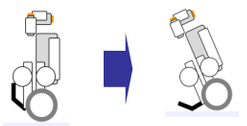
コースは区間を持ち、区間はサブ区間を持つ。サブ区間は”走法”と”終了条件” を持っており、終了条件を満たすまで、ひとつの走行を続けることで攻略される。コース全体はこの基本シーケンスの連続により攻略できる。

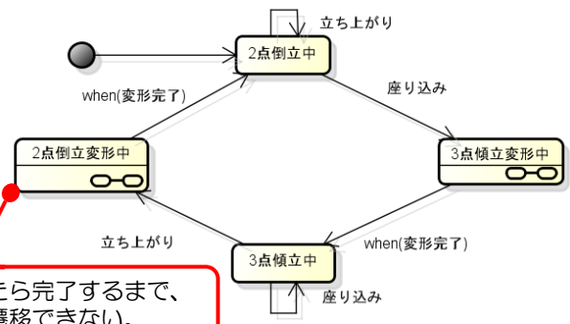


最後のサブ区間が終了した際は次の区間をコースに返す

2.2 姿勢の状態遷移

走行体が持つ”現在姿勢状態”の状態遷移を記載する。



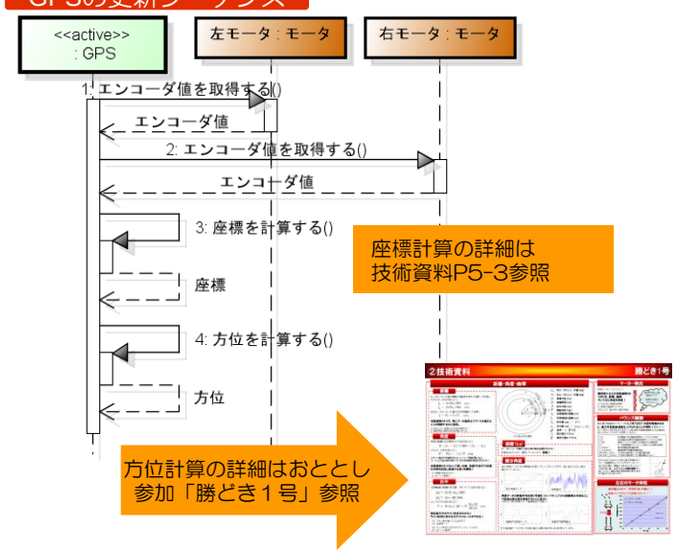


変形し始めたら完了するまで、他の状態に遷移できない。

3 GPSタスク

モータから取得したエンコーダ値を基にクラス内部で座標、方位情報が計算されている。終了条件としてこれらが利用される。

GPSの更新シーケンス



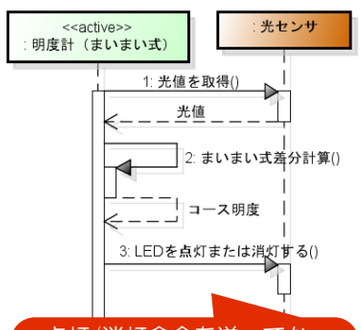
座標計算の詳細は技術資料P5-3参照

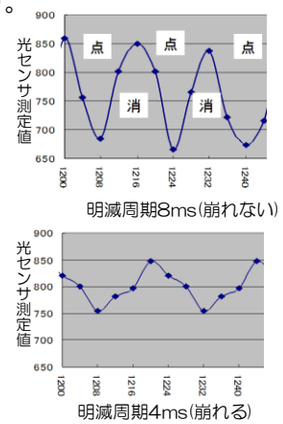
方位計算の詳細はおとなし参加「勝どき１号」参照

4 まいまいタスク

LEDの明滅を繰り返し、外乱光に強いセンシングをおこなう。オリジナルのまいまい式で行われていた複数タスクの生成、廃棄の無駄を排除し、タスクを１本化することで、センシングの周期を20msから10msに改善することができた(東京連合式)。

まいまい式明度計





点灯/消灯命令を送ってから、きちんとLEDが点灯/消灯されるまでハードウェアの制約により約8ms待つ必要がある。

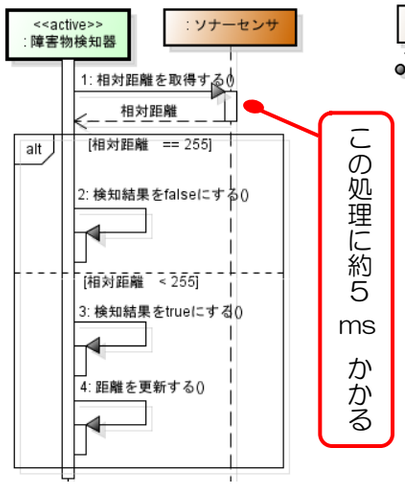
※東京連合大西さん発表資料より

まいまい式(改)については技術資料P5-5にも記載

5 障害物検知タスク

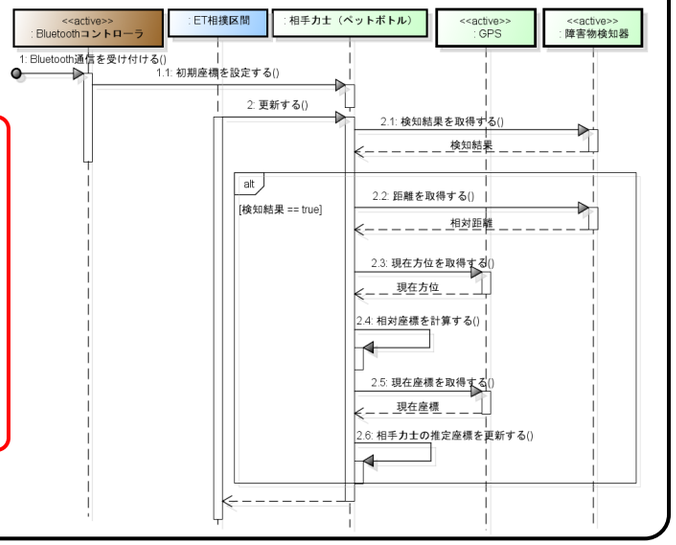
相手力士（ペットボトル）を検知し、推定座標等を更新する。
ソナーセンサを利用するため実行時間が長い。そのため独立したタスクとしている。
障害物検知器およびその利用シーケンスを記載する。

障害物検知器



この処理に約５ms かかる

利用例（ペットボトルの推定座標更新シーケンス）



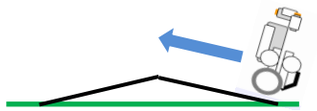
1 坂道走行

平地で走っている速度で、坂道を走行すると下り坂で、加速し過ぎで転倒する。
進行速度を目標値とした“速度PID制御”を導入することで、上り・下りで最適なフォワード値を自動制御する。

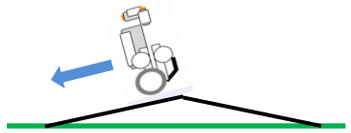
手法	成功率	考察
1．定フォワード値	0/10	速度PIDと同速度で坂道を攻略しようとすると、下り坂で転倒。
2．速度PID	10/10	下り坂に突入時、適切に減速し転倒回避。

採用

上りでは力強く
（大きなフォワード値）

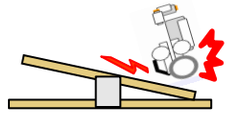


下りでは慎重に
（小さなフォワード値）



2 ダブルクラッチ

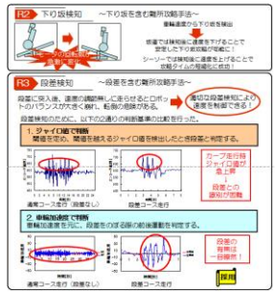
ジャイロ・オフセット値を大きくすると急発進でスタートし、オフセット値を小さくすると、急ブレーキを行う。ダブルクラッチ攻略時にオフセット値を変更し、急ブレーキすることで、シーソー上に留まる。



手法	成功率	考察
1．下り坂検知 ＋　ジャイロ・オフセット	0/10	下り坂検知後に、急ブレーキを掛けても、シーソーから落ちてしまう。
2．下り坂検知 ＋　マイナス・フォワード値	0/10	下り坂検知後に、急ブレーキを掛けても、シーソーから落ちてしまう。
3．段差検知　＋　走行距離 ＋　ジャイロ・オフセット	7/10	下り坂検知後にブレーキを掛けても手遅れであるため、実際にシーソーが傾く手前でブレーキを掛ける。シーソーが傾くタイミングの推測は段差検知＋走行距離で行う。
4．段差検知　＋　走行距離 ＋　マイナス・フォワード値	5/10	ジャイロ・オフセットの変更より、ブレーキの力が弱いため、成功確率が低い。

採用

下り坂検知・段差検知に関しては、２０１０年勝ロボ・チャンピオンシップ大会のモデルを参照。



3 ルックアップ・ゲート

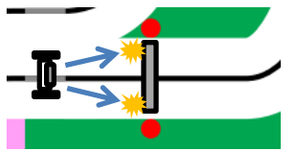
走行体が、ルックアップ・ゲート・エリアの座標に到達*1した時点から、ルックアップ・ゲートの攻略を開始する。エリアに突入した時点で、走行体は、姿勢制御*2を行い３点傾立状態になる。
３点傾立状態で、方位トレース、または、ライントレースを行い、ゲートを通過する。*3
ゲートの通過判定は、ルックアップ・ゲート・エリアに突入してからの走行距離*4を利用して行う。

手法	成功率	考察
1．方位トレース	8/10	エンコーダ値の誤差で、進行方向にばらつきが生じ、ゲートにクラッシュすることあり。
2．ライントレース	10/10	エンコーダ値の誤差に影響されず、ゲートを通過することができる。

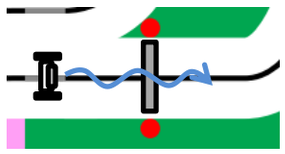
採用

*1　走行体の現在位置の測定については、P.5-4を参照のこと。
*2　姿勢制御については、P.5-2を参照のこと。
*3　３点傾立ライントレースは、企業秘密、ゴメンナサイ。
*4　走行距離の測定については、P.5-4を参照のこと。

手法　その１



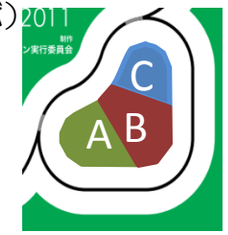
手法　その２



4 ET相撲　～確実性重視～

4.1 確実なペットボトル発見（Bluetooth通信機器とのコラボ）

ペットボトル配置エリアを３つのサブエリアに分割。スタート直前にBluetooth通信機器へ入力した情報をもとに、通信機器から走行体へ、ペットボトルの配置されているサブエリアを通知。
走行体は、指定されたサブエリアを重点的に探す。

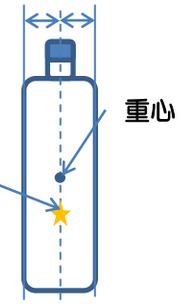


4.2 確実な押し出し

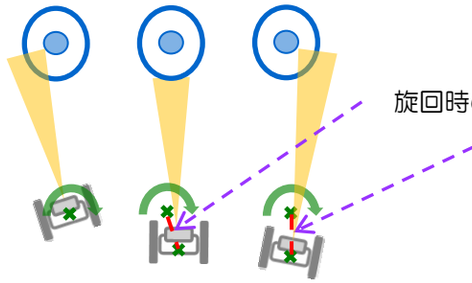
押し倒し(ボーナスタイム１５秒)ではなく、**押し出し**（ボーナスタイム２５秒）を成功させるためにはスウィート・スポットを押す。

- ペットボトルの左右の中心
- 重心より低い位置

} スウィート・スポット

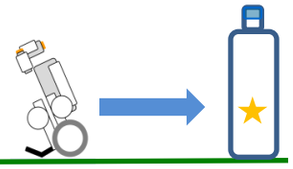


ペットボトルの中心を検知
旋回を行いながら、超音波センサによりペットボトルの中央を探す。超音波センサの値（ペットボトルまでの距離）を直接使用すると、旋回のための誤差（下図）がでる。誤差を減らすために、走行体のGPS機能（走行体の現在位置、進行方向、P.5の４を参照）を利用して、ペットボトルの座標を計算。求めた座標に向けて、方位トレース走法で、ペットボトルを押し出す。

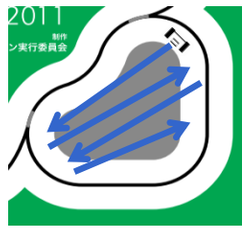


旋回時の滑りによる誤差

理論上では、その場で旋回してペットボトルを検索した場合、ペットボトルの中央の方向で、ペットボトルまでの距離が最短になるはず。しかし、旋回時の滑りによる誤差で、実際には、ペットボトルの中央の方向で、ペットボトルまでの距離が最短にならない。



３点傾立走行で、ペットボトルの重心より低い位置を押す。

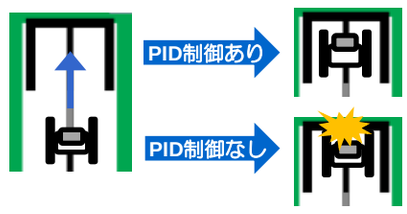


4.3 それでも、ダメな場合には・・・

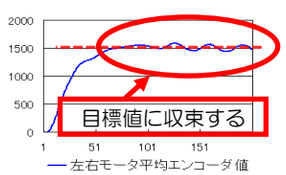
もし、ペットボトルが押し出せなかった場合は、ET相撲のエリアを隙間なくスウィープ！！

5 ガレージ・イン

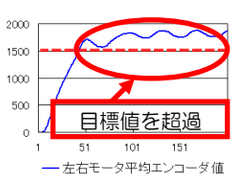
ガレージ・インは、座標指定走法によりガレージ中央に正確に止まることで攻略する。



座標指定走法
目標座標（ガレージ・イン停止位置）で、正確に停止するために、走行体の現在位置と、目標座標までの距離からフォワード値を算出（PID制御）。目標座標に近づくほど、走行速度を落とすため、オーバランせずに正確に停止することができる。



PID制御あり



PID制御なし

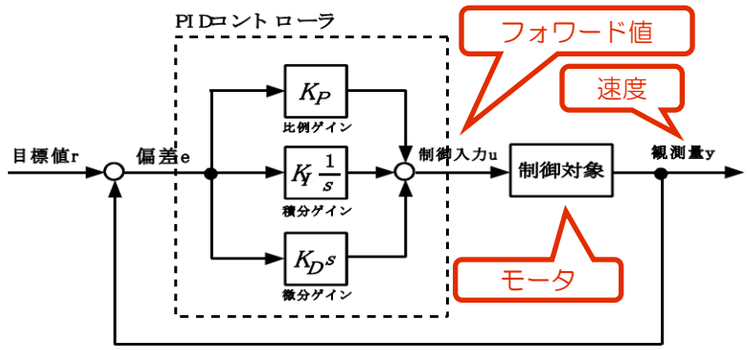
5 技術資料

要素技術の解説

1 速度PID制御

坂道でもカーブでも、定速度の安定した走行が可能！

モータエンコーダ値を観測し、実際の速度からフォワード値を調節する。
パラメータ決定には**限界感度法**を利用した。

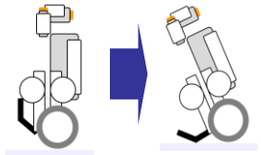


2 姿勢制御

2.1 座り込み

手法	成功率	考察
1. 尻尾を下げ、バランス制御を切り、一時的に速度をあげる。	10/10	特に困難がなかったため、他の手法は試さなかった。

速度をあげ体をのけぞらす

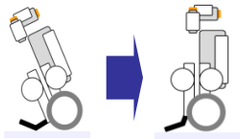


2点直立　3点傾立

2.2 立ち上がり

手法	成功率	考察
1. 尻尾をP制御で押し下げることで体を起こす。	0/10	2011年サンプルコードのtail_controlを利用。パワーが足りず車体が持ち上がらなかった。
2. 尻尾をP制御で押し下げると同時に、車輪を後転することで体を起こす。	5/10	成功率５割。勢い余って前に倒れることがある。また、尻尾と左右車輪の３つを制御する必要があるため、調節が難しかった。
3. 尻尾に直接PWM値を指定して押し下げることで体を起こす。	9/10	尻尾だけで体を持ち上げることができた。尻尾だけの制御ですみ、調節が楽。

尻尾で体を持ち上げる



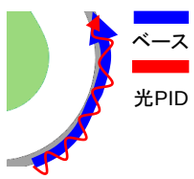
3点傾立　3点直立

採用

3 ライントレース高速化

レーン形状を考慮したライントレース！

レーンを分割し、分割区間に最適なベース旋回量をあらかじめ設定しておくことにより、走行タイムを向上させると共に安定した走行を実現できた。

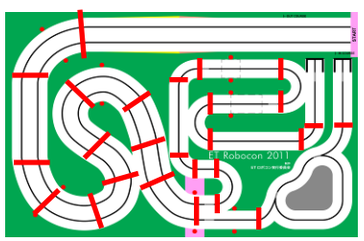


ベース旋回量を基準値として、　**光PIDによる調節を行う**

旋回量 = ベース旋回量(座標) + 光PID制御(光値)

レーン分割

走行体を走らせ、レーンの形状(曲率)を計測。曲率を元に、教師なし機械学習を用いて、レーンを自動分割した。分割個数は任意。



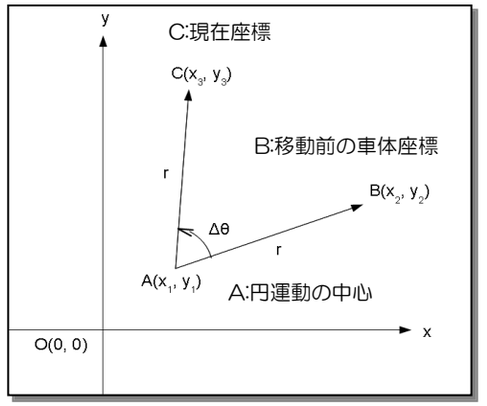
4 GPS　～自己位置推定手法～

4.1 座標推定

左右モータのエンコーダ値から、スタート地点からの相対座標、向き変化を求めることで、GPS機能を実現する。

現在座標Cの座標は以下の式で表される。

$$x_3 = \cos\Delta\theta(x_2 - x_1) - \sin\Delta\theta(y_2 - y_1) + x_1$$
$$y_3 = \sin\Delta\theta(x_2 - x_1) + \cos\Delta\theta(y_2 - y_1) + y_1$$



4.2 誤差補正

GPSは、進行方向がスタート時の向きに依存し、推定した現在座標と実際の座標が大きく異なってしまう場合がある。そのため、直線コース走行時の進行方向補正を導入した。

丸印の区間を走行しているときに進行方向の補正を行う

スタート時に向きを付けた場合、補正を導入することにより劇的な効果が得られた。

※画像は2010年コースのものです

5 まいまい式(改)　～外乱光対策～

自然光が入る東京地区大会では必須！？外乱光に強いセンシング手法。
- 「LED点灯時の測定値」と「LED消灯時の測定値」の差分を計算すると、
- 「LEDの反射光成分」だけが残る（外乱光の影響をキャンセルできる）、
- というのが「まいまい式」の原理です。※すねいるさんブログより引用

欠点：倒立制御の周期（4ms）に対し、センシングの周期が20msと長い。
改良：タスクの生成、破棄の無駄を排除し、センシングの周期を10msに改善した（東京連合式）。