

1. 要求分析

選択課題: コースを完走する

チーム目標: 制限時間内にコースを完走する

私たちチーム「ハナ☆花ライダー」は昨年度の先輩方の結果(Rコースのみ完走)を振り返り、「必ずゴールに到達する」という意志の元、要求分析を行いました。

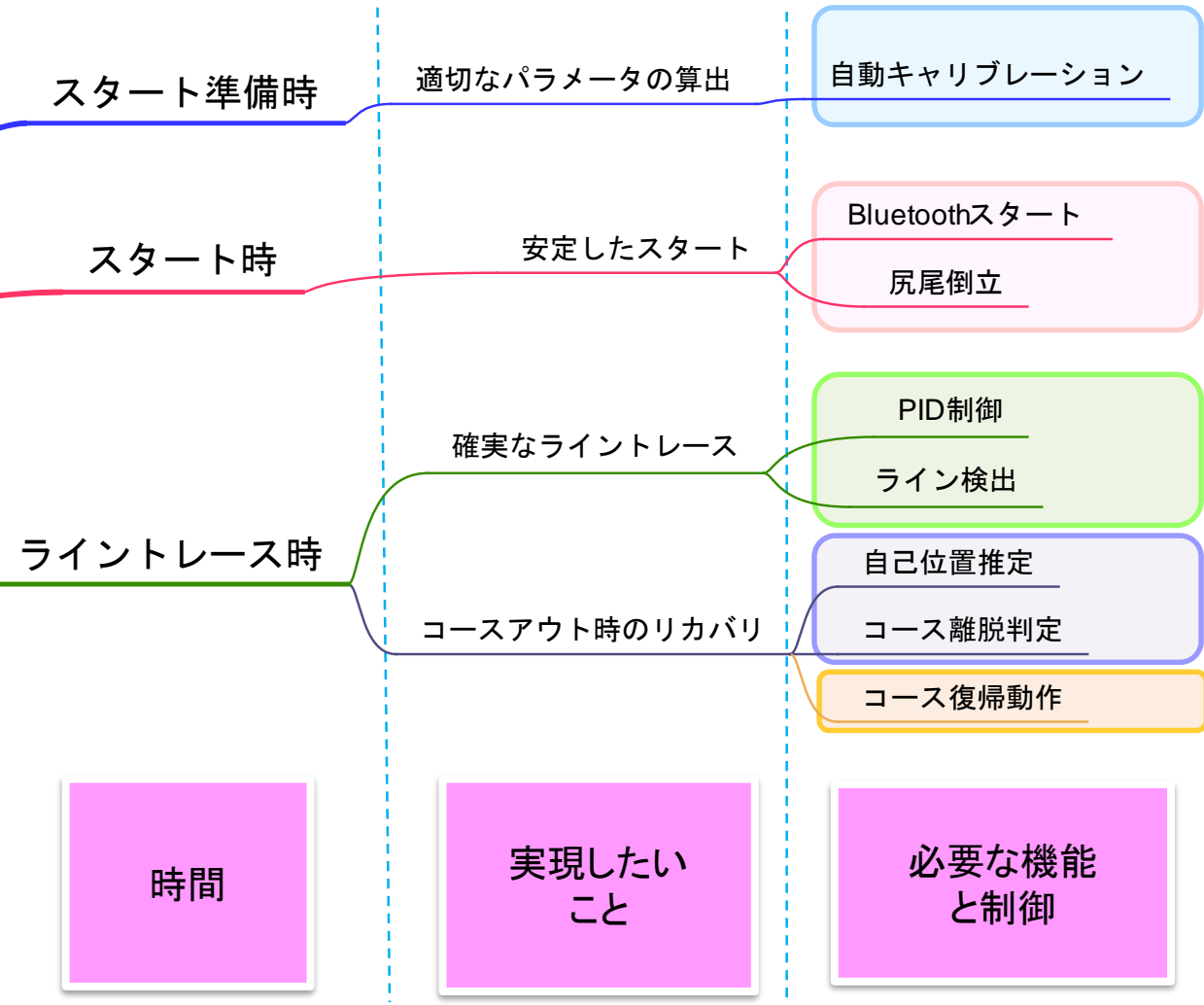
コース環境の影響によるコース逸脱を判定(コース離脱判定)し、その後にコース復帰動作を盛り込むことで、私たちは何としてもベーシックコースを走り抜けます。

その他、ライントレース時と同じ環境で黒色と白色の光センサ値を取得できるように、二輪倒立を行いながらキャリブレーションを行う。スタート時は、人の手を介するスタートは安定性が失われるため、尻尾で走行体を倒立させて、Bluetoothを用いてリモートスタートを行う。

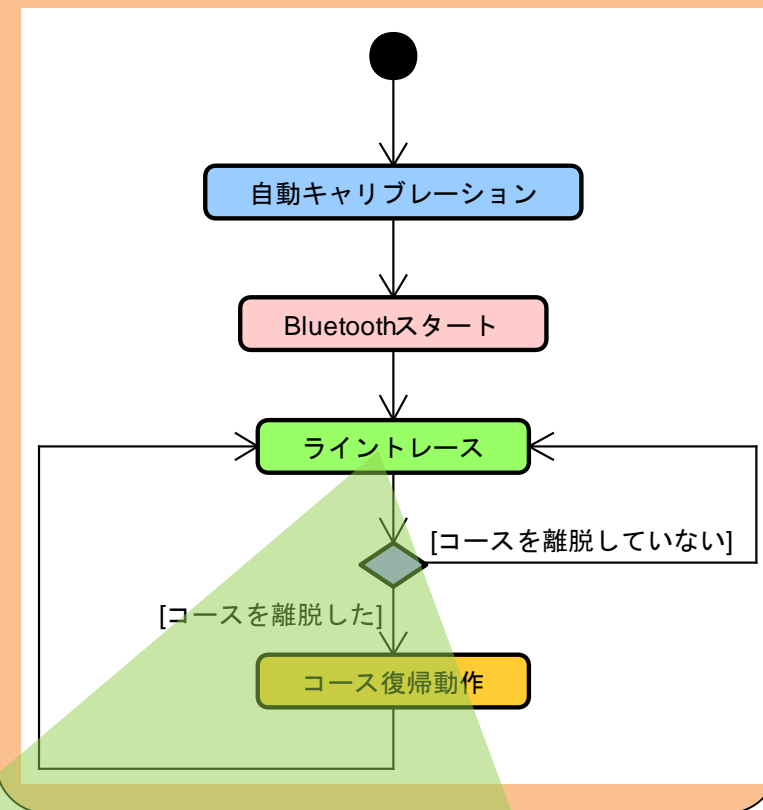
右の全体アクティビティ図では、要求分析で抽出された機能と制御の処理フローを示した。右下のアクティビティ図はコースを完走するために必要なライントレースの詳細を示している。

要求分析

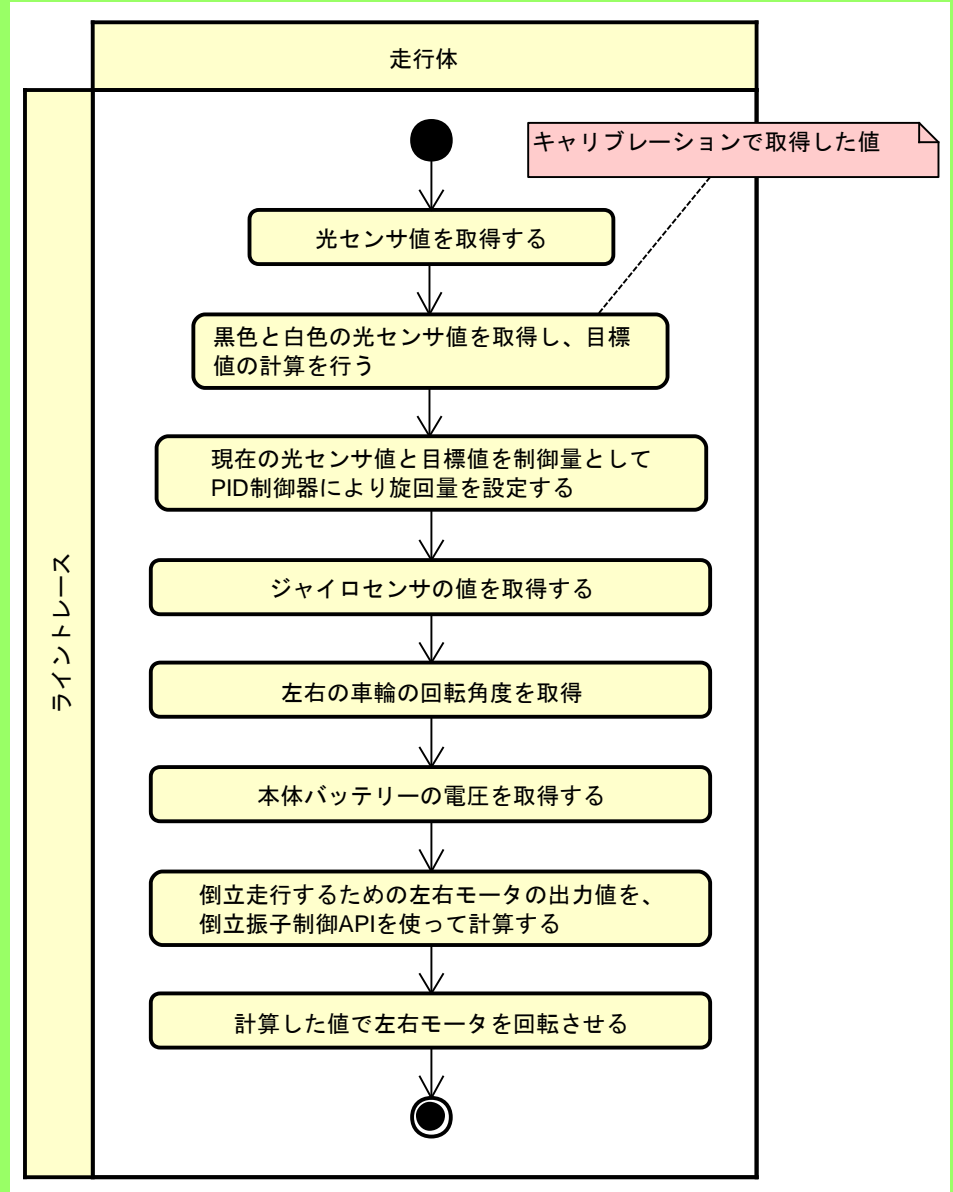
コースを完走する



全体アクティビティ図



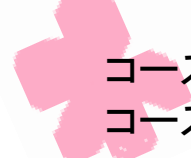
ライントレース



コー

コー

コー



全体アクティビティ図

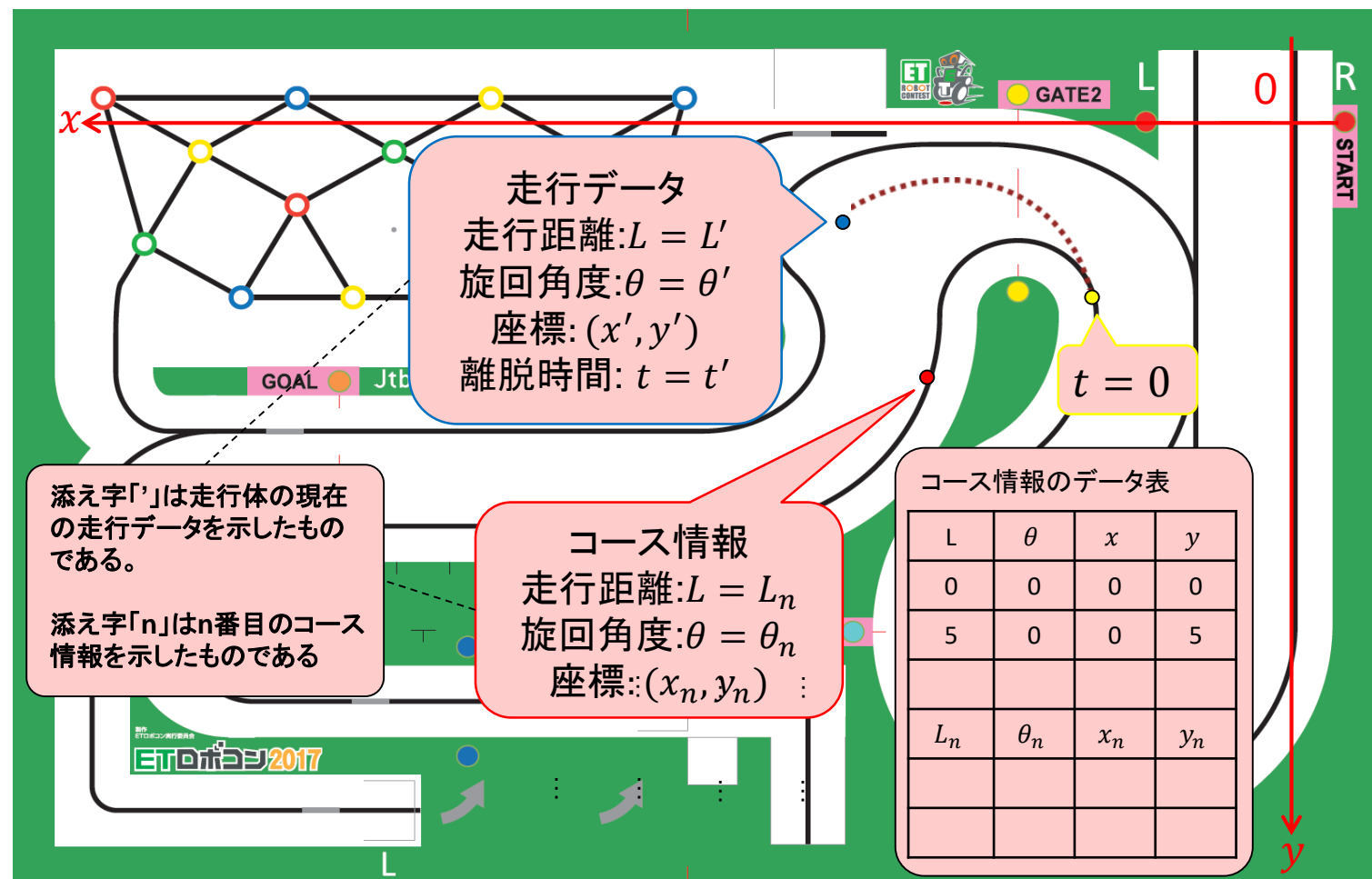
アクティビティ図は、コースを完走するために必要なライトレースと、私たちのアピールポイントのコース復帰制御の機能に関する記載に絞った。

そのため自動キャリブレーションとBluetoothスタートのアクティビティ図は省略させていただいた。

3. 復帰制御の実現方法

次に、コース離脱判定とコース復帰動作それぞれの実現方法について記載する。

コース離脱判定 走行体は常に自己位置推定を行っている。ここではその自己位置の情報を走行データと呼ぶ。走行データとコース情報を比較することで離脱の判定を実現する。



記号	名称	単位	意味
L	走行距離	cm	スタートからの走行した距離
θ	旋回角度	度	スタート時の向きを基準にした走行体の向き
(x, y)	座標	cm	スタート位置を原点とした走行体の位置
t	離脱時間	秒	ラインを最後に検知してからの経過時間

離脱条件

$|L_n - L'| \leq 2.5[\text{cm}]$ であるとき
 $t' \geq 3[\text{秒}]$ かつ
 $|\theta_n - \theta'| \geq 30[\text{度}]$ かつ
 $\sqrt{(x_n - x')^2 + (y_n - y')^2} \geq 15[\text{cm}]$

最後にラインを検出してから3秒以上経過した場合に、走行体の走行距離に応じたコース情報を取得する。現在の走行データと取得したコース情報を比較する。旋回角度のズレが30度以上で、座標のズレが15cm以上の場合に離脱と判定する。

コース復帰動作 離脱と判定された場合、走行体は正しいコースに復帰する必要がある。走行データとコース情報を用いて、最適なコース復帰を実現する。

①目標座標の方向へ旋回する

離脱判定で取得したコース情報の座標を目標座標とする。現在の走行体の座標と目標座標から、目標座標へ向かうための旋回角度 φ を計算する。

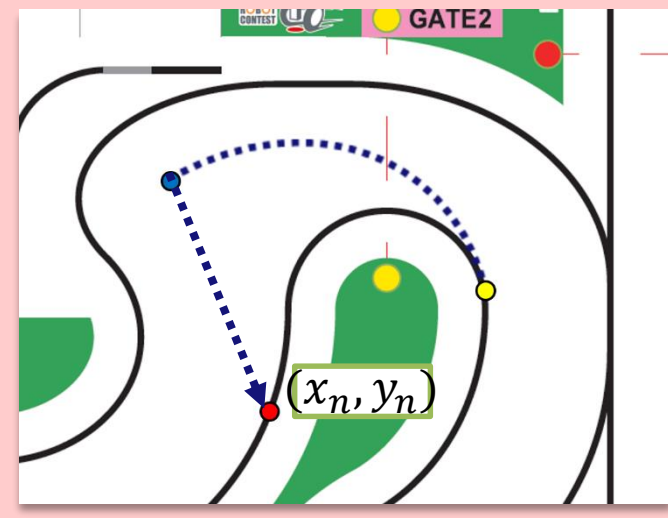
$$\varphi = \tan^{-1} \frac{x_n - x'}{y_n - y'}$$

走行体は $(\varphi - \theta')$ だけ旋回して、目標座標の方向を向く。



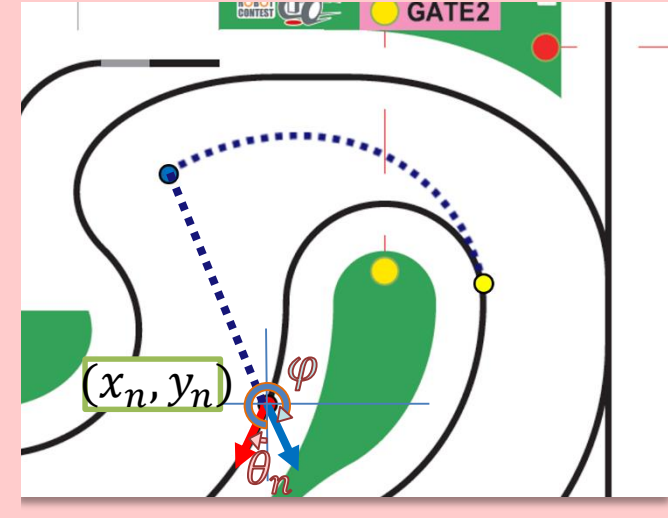
②目標座標の方向へ直進する

光センサを起動し、反射光の強さを計測しながら目標座標へ向かって直進する。
光センサがラインを検知したら一時停止する。



③目標旋回角度まで旋回する

離脱判定で取得したコース情報の旋回角度を目標旋回角度とする。②でラインを検知したときの走行体の旋回角度は φ である。
走行体は $(\theta_n - \varphi)$ だけ旋回して、コースに沿った方向を向く。さらに走行データを、離脱判定で取得したコース情報に書き換える。



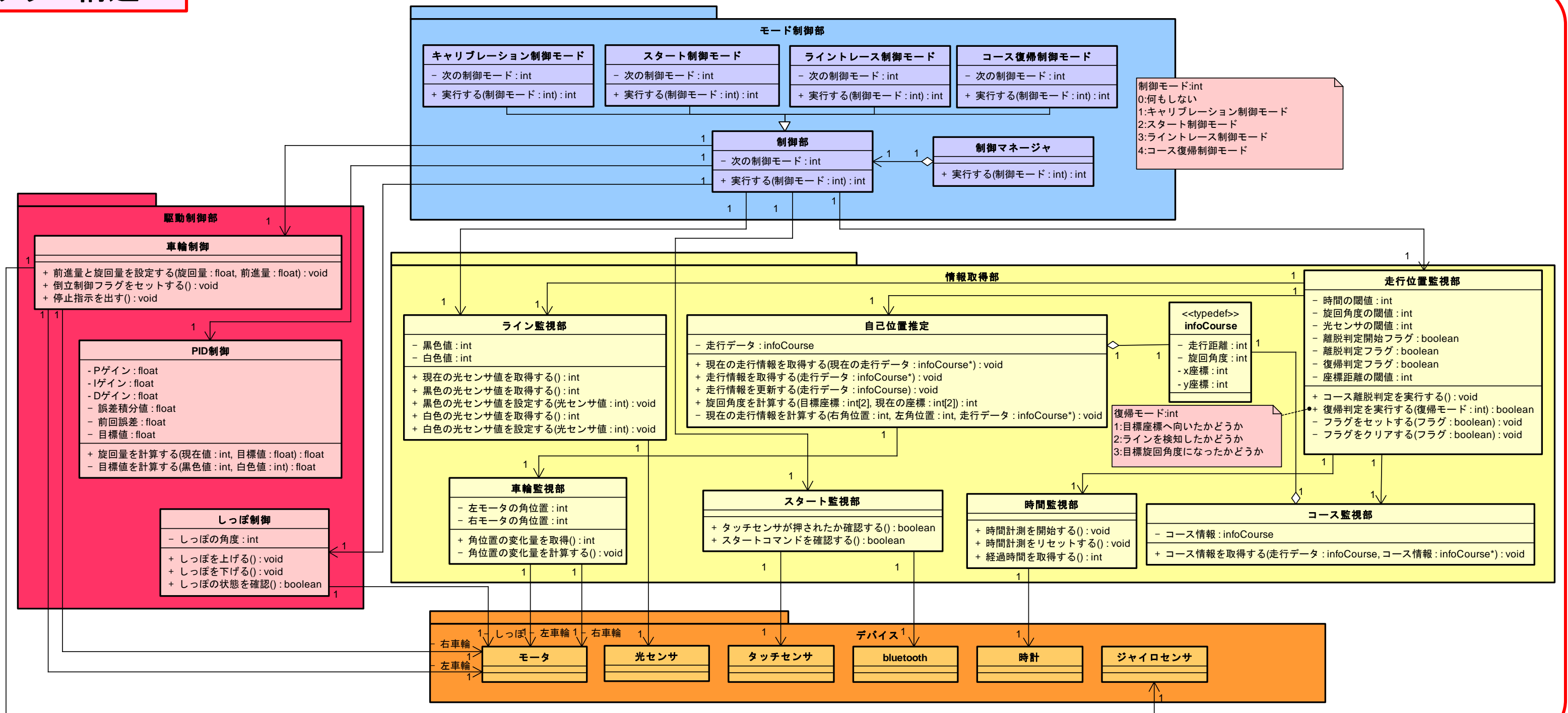
4. 構造分析

次に、全体のクラス構造を示す。
クラスのパッケージとして、モード制御部、駆動制御部、情報取得部、デバイスの大きく4つに分けられる。

パッケージ説明

パッケージ名	役割
モード制御部	各モードの制御を行う
駆動制御部	走行体の駆動系の制御を行う
情報取得部	デバイスや所持データなどから情報を取得する
デバイス	走行体のモータやセンサなどのAPIを呼び出す

クラス構造



「4. シー

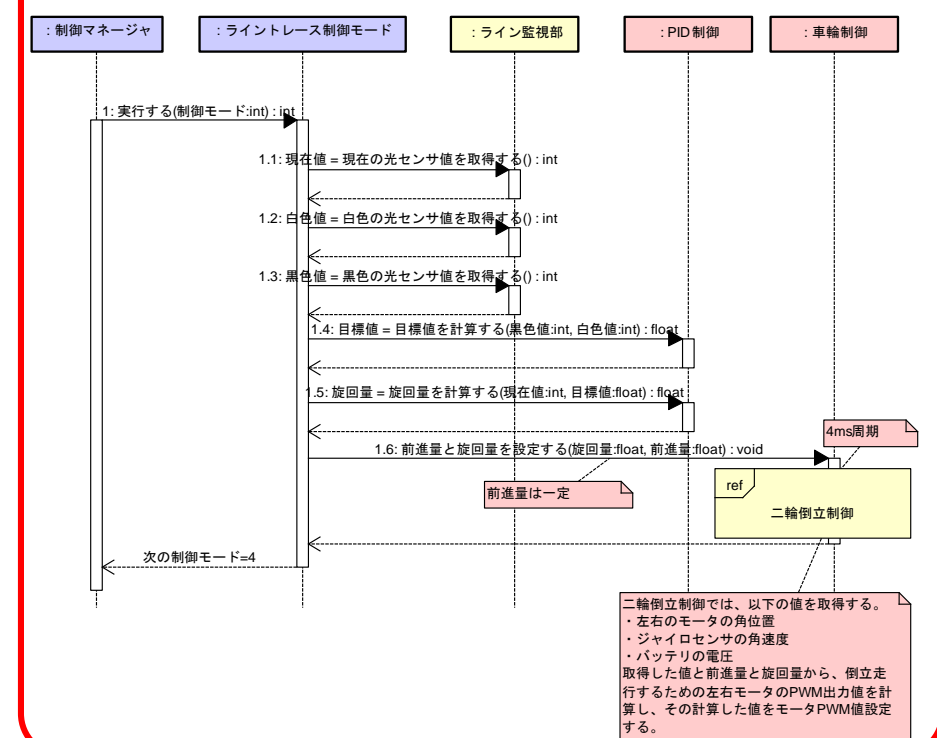
ここではコースを完走するために必要なライントレースと、私たちのアピールポイントのコース復帰制御のシーケンス図の記載だけに限らせていただいた。自動キャリブレーション、Bluetoothスタートについては省略した。

```
sequenceDiagram
    participant Manager as : 制御マネージャ
    participant Controller as : 制御部
    participant Modes as 各制御モード
    Manager->>Controller: 1: 実行する(制御モード: int)
    activate Controller
    Controller->>Modes: ref
    activate Modes
    Modes-->>Controller: 次の制御モード
    deactivate Modes
    Controller-->>Manager: 
    deactivate Controller
```

The diagram illustrates the control mode management process. It features three lifelines: **: 制御マネージャ** (Control Manager), **: 制御部** (Control Unit), and **各制御モード** (Each Control Mode). The process begins with the Control Manager sending a message **1: 実行する(制御モード: int)** to the Control Unit. The Control Unit then sends a **ref** message to the **各制御モード** object. The **各制御モード** object returns **次の制御モード** (Next Control Mode) to the Control Unit. Finally, the Control Unit returns the result to the Control Manager. A **loop** box encloses the interaction between the Control Manager and the Control Unit.

②目標座標の方向
へ直進する
に対応する

ライトレースの処理フロー



```

sequenceDiagram
    participant 制御部
    participant コース復帰制御モード as : コース復帰制御モード
    participant 走行位置監視部 as : 走行位置監視部
    participant 自己位置推定 as : 自己位置推定
    participant 車輪制御 as : 車輪制御
    participant コース監視部 as : コース監視部
    participant ライン監視部 as : ライン監視部

    制御部->>1: 実行する()
    activate 制御部

    1.1: 離脱判定フラグ = コース離脱判定を実行する() : void
    activate コース復帰制御モード
    コース復帰制御モード->>ref: ref  
コース離脱判定
    deactivate コース復帰制御モード

    1.2: 停止指示を出す() : void
    activate 制御部
    deactivate 制御部

    loop
        1.3: 前進量と旋回量を設定する(旋回量:float, 前進量:float) : void
        activate 制御部
        旋回量=0  
前進量=0  
走行体を回転させる:
        activate 車輪制御
        ref: ref  
二輪倒立制御
        4ms周期
        deactivate 車輪制御
        deactivate 制御部

        1.4: 復帰判定フラグ = 復帰判定を実行する(復帰モード=1) : boolean
        activate コース復帰制御モード
        1.4.1: 現在の走行情報を取得する(現在の走行データ:infoCourse*) : void
        activate 自己位置推定
        ref: ref  
自己位置推定
        deactivate 自己位置推定
        1.4.2: コース情報を取得する(走行データ:infoCourse, コース情報:infoCourse*) : void
        activate コース監視部
        ref: ref  
コース監視部
        deactivate コース監視部
        1.4.3: 旋回角度 = 旋回角度を計算する(目標座標:ref[2], 現在の座標:int[2]) : int
        activate 走行位置監視部
        ref: ref  
走行位置監視部
        deactivate 走行位置監視部
        opt [旋回角度の現在値と計算値の差が5度以内]
            1.4.4: フラグをセットする(復帰判定フラグ:boolean) : void
            activate コース復帰制御モード
            deactivate コース復帰制御モード
        end
        deactivate コース復帰制御モード

        break [復帰判定フラグ==True]
        1.5: 停止指示を出す() : void
        activate 制御部
        deactivate 制御部
    end

    loop
        1.6: 前進量と旋回量を設定する(旋回量:float, 前進量:float) : void
        activate 制御部
        旋回量=0  
前進量=0  
走行体を直線移動する:
        activate 車輪制御
        ref: ref  
二輪倒立制御
        4ms周期
        deactivate 車輪制御
        deactivate 制御部

        1.7: 復帰判定フラグ = 復帰判定を実行する(復帰モード=2) : boolean
        activate コース復帰制御モード
        1.7.1: 光センサの現在値 = 現在の光センサ値を取得する() : void
        activate ライン監視部
        ref: ref  
ライン監視部
        deactivate ライン監視部
        opt [光センサ値<光センサの閾値]
            1.7.2: フラグをクリアする(復帰判定フラグ:boolean) : void
            activate コース復帰制御モード
            deactivate コース復帰制御モード
        end
        deactivate コース復帰制御モード

        break [復帰判定フラグ==True]
        1.8: 停止指示を出す() : void
        activate 制御部
        deactivate 制御部
    end

    loop
        1.9: 前進量と旋回量を設定する(旋回量:float, 前進量:float) : void
        activate 制御部
        旋回量>0  
前進量=0  
走行体を回転させる:
        activate 車輪制御
        ref: ref  
二輪倒立制御
        4ms周期
        deactivate 車輪制御
        deactivate 制御部

        1.10: 離脱判定フラグ = コース離脱判定を実行する(復帰モード=3) : void
        activate コース復帰制御モード
        1.10.1: 現在の走行情報を取得する(現在の走行データ:infoCourse*) : void
        activate 自己位置推定
        ref: ref  
自己位置推定
        deactivate 自己位置推定
        1.10.2: コース情報を取得する(走行データ:infoCourse, コース情報:infoCourse*) : void
        activate コース監視部
        ref: ref  
コース監視部
        deactivate コース監視部
        opt [旋回角度の現在値とコース情報の差が5度以内]
            1.10.3: フラグをクリアする(離脱判定フラグ:boolean) : void
            activate コース復帰制御モード
            deactivate コース復帰制御モード
            1.10.4: 走行情報を更新する(走行データ:infoCourse) : void
            activate 走行位置監視部
            ref: ref  
走行位置監視部
            deactivate 走行位置監視部
        end
        deactivate コース復帰制御モード

        break [離脱判定フラグ==False]
    end

    制御部->>2: 次の制御モード=3
    deactivate 制御部
  
```

```
sequenceDiagram
    participant Self as : 自己位置推定
    participant Wheel as : 車輪監視部

    Self->>Wheel: 1: 左車輪 = 角位置の変化量を取得(): int
    activate Wheel
    Wheel->>Wheel: 1.1: 角位置の変化量を計算する(): void
    deactivate Wheel
    Wheel-->>Self: 
    Self->>Wheel: 2: 右車輪 = 角位置の変化量を取得(): int
    activate Wheel
    Wheel->>Wheel: 2.1: 角位置の変化量を計算する(): void
    deactivate Wheel
    Wheel-->>Self: 
    Self->>Self: 3: 現在の走行情報を計算する(右角位置: int, 左角位置: int, 走行データ: infoCourse*) : void
    deactivate Self
```

```

sequenceDiagram
    participant W as : 走行位置監視部
    participant S as : 自己位置推定
    participant T as : 時間監視部
    participant L as : ライン監視部
    participant C as : コース監視部

    W->>S: 1: 現在の走行情報を取得する(現在の走行データ:infoCourse*) : void
    activate S
    S-->>W: 
    deactivate S
    Note right of S: 現在の走行距離、旋回角度、座標を取得  
→ 走行距離の現在値  
→ 旋回角度の現在値  
→ 座標の現在値と定義
    W->>W: opt  
[離脱判定開始フラグ==False]
    W->>S: 2: フラグをセットする(離脱判定開始フラグ) : void
    activate S
    S-->>W: 
    deactivate S
    W->>T: 3: 時間計測を開始する()
    activate T
    T-->>W: 
    deactivate T
    W->>L: 4: 現在の光センサ値を取得する() : int
    activate L
    L-->>W: 
    deactivate L
    Note right of L: 走行距離に対応した座標と旋回角度を取得  
→ 旋回角度の目標値  
→ 座標の目標値と定義
    W->>W: alt : 光センサ値
    W->>T: 5: 経過時間を取得する() : int
    activate T
    T-->>W: 
    deactivate T
    W->>W: opt : 経過時間
    W->>C: 6: コース情報を取得する(走行データ:infoCourse, コース情報:infoCourse*) : void
    activate C
    C-->>W: 
    deactivate C
    W->>W: opt  
[旋回角度の目標値と現在値の差 > 旋回角度の閾値]
    W->>W: opt  
[座標の目標値と現在値の差 > 座標距離の閾値]
    W->>W: 7: フラグをセットする(離脱判定フラグ) : void
    activate W
    W-->>W: 
    deactivate W
    W->>T: 8: 時間計測をリセットする()
    activate T
    T-->>W: 
    deactivate T
    Note right of T: ライン上
    W->>W: 
    deactivate W
  
```