開発環境・要素技術セミナー

ETロボコン2016

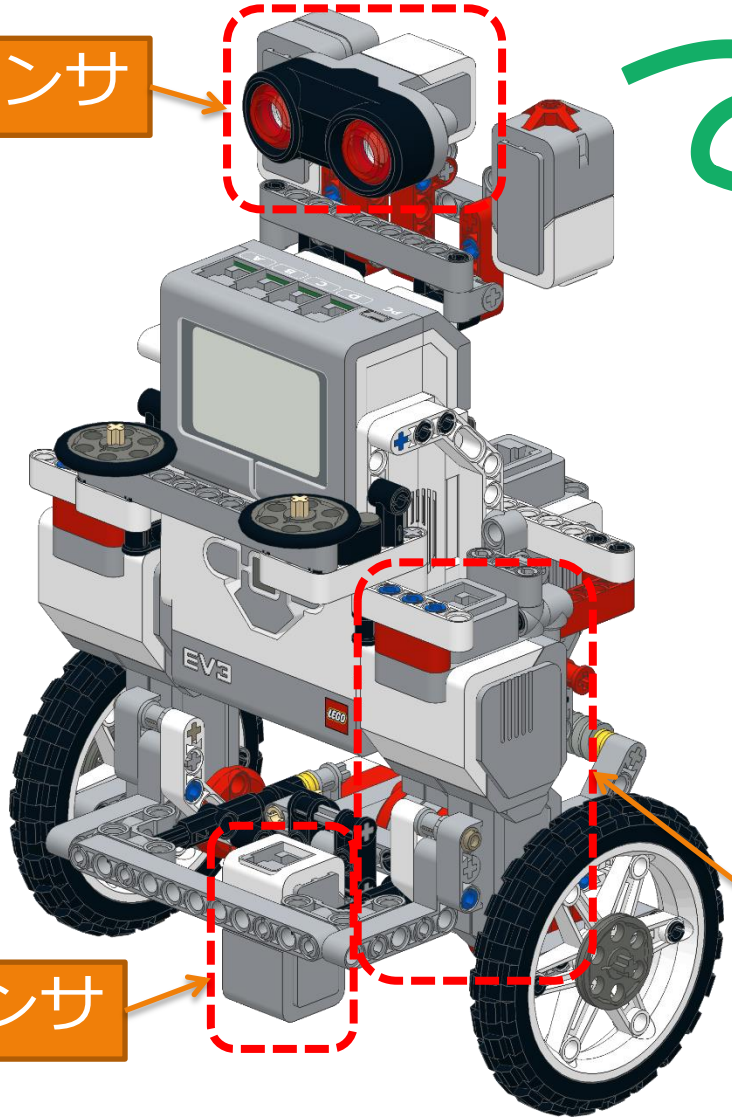
目次

- ハードウェアの特性
- 倒立振子ライブラリ
- EV3の開発環境
- RTOS
- EV3のデバイスAPI
- お願い

ハードウェアの特性

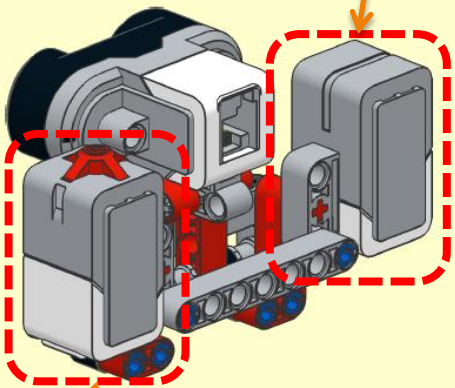
まずは走行体、これを動かすのが目的です
走行体がどんな部品で出来ているか確認します

走行体に繋っているもの



超音波センサ

ジャイロセンサ



タッチセンサ

※本章では扱いません

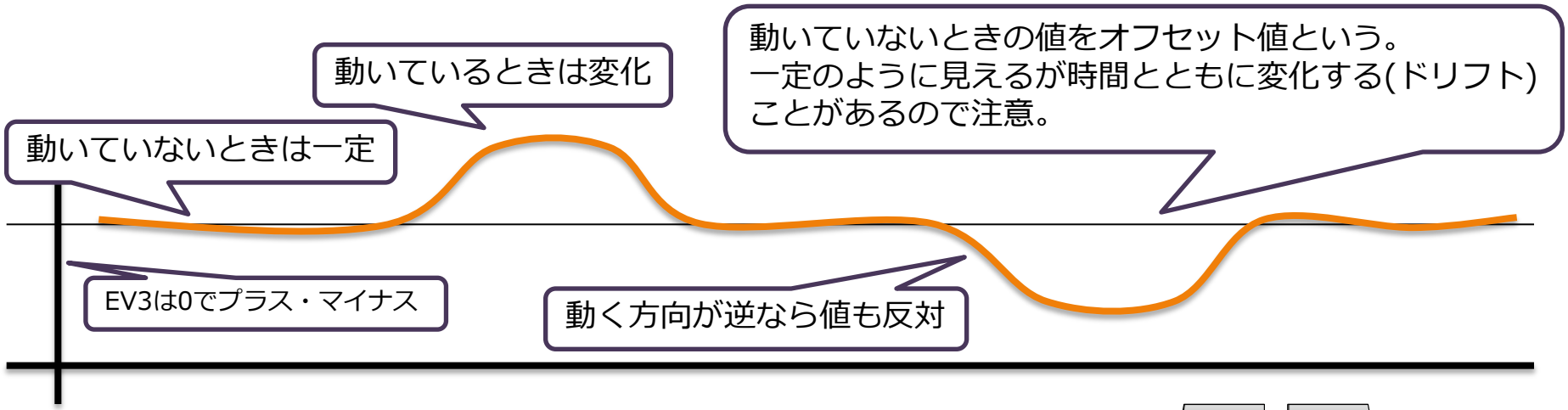
モータ

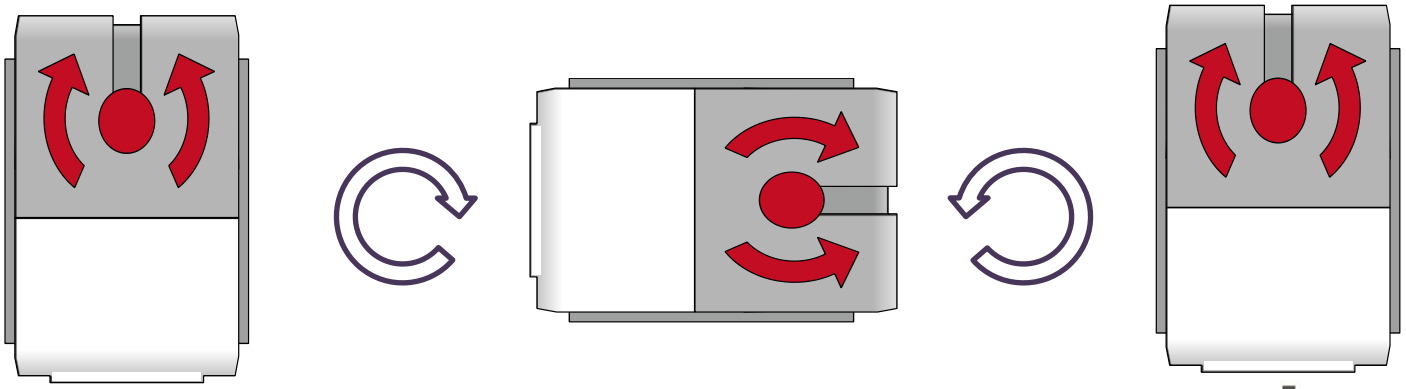
※エンコーダ付き

カラーセンサ

ジャイロセンサ

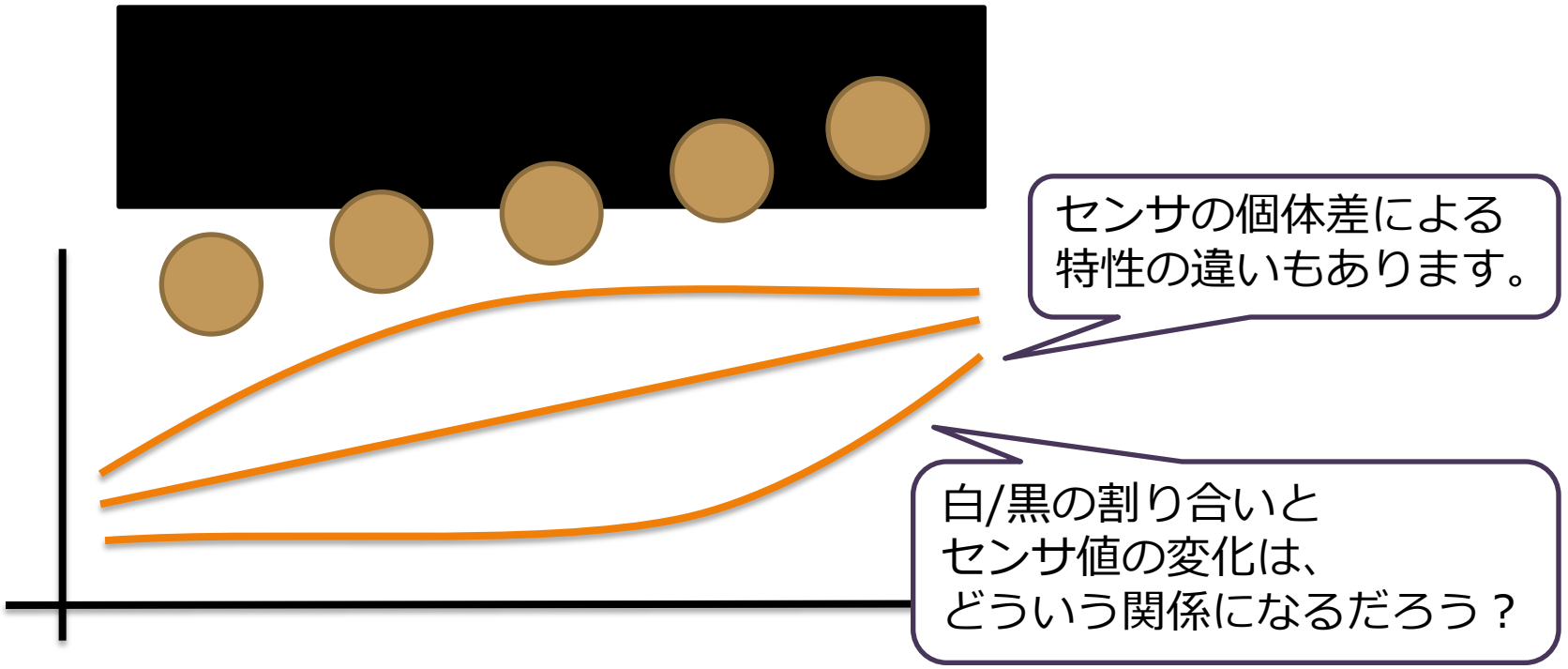
- **角速度**を計測するセンサ
  - センサの置かれている角度を計るものではありません。
  - ある時から、ある時までの動きを検出します。





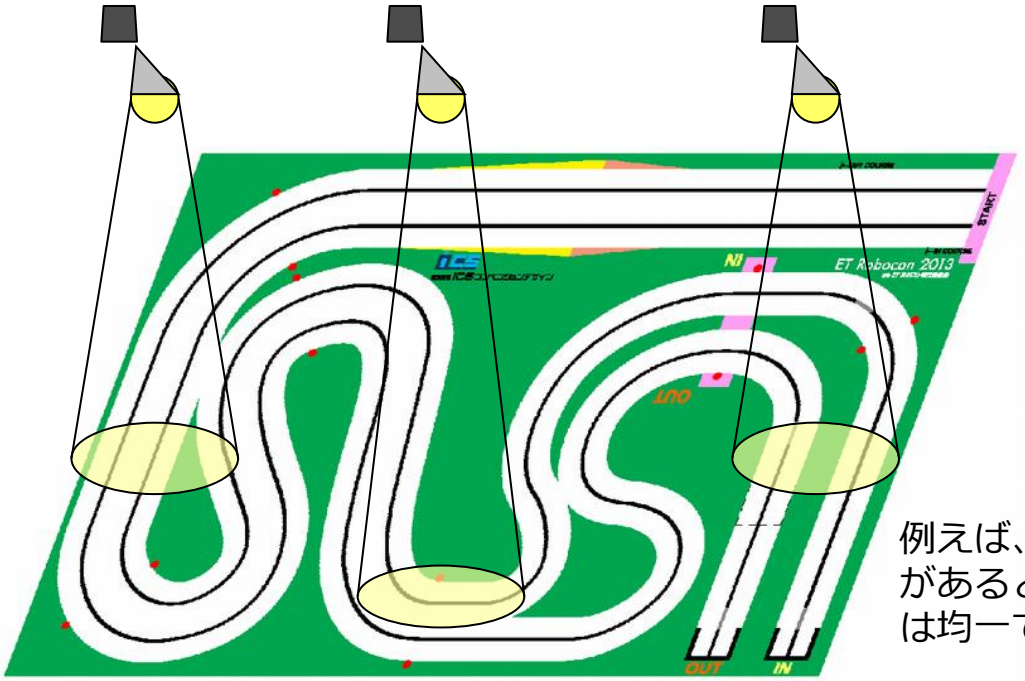
カラーセンサ（反射光検出モード）

- 発光して反射した光で明るさを検出します。
- 明るさとセンサ値の関係
  - 明るさとセンサ値の関係は
    リニア(単純な比例関係)ではありません。



外乱の影響

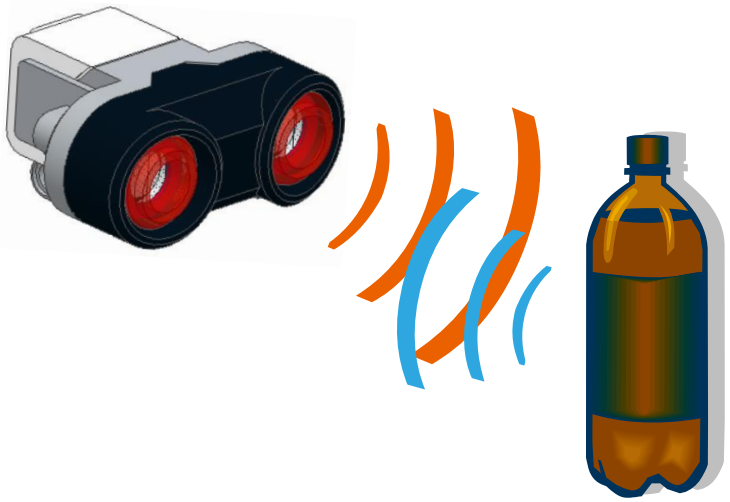
- コースの白地と黒線は、一様なRGB値で作成されています。しかしカラーセンサから取得した明るさは、コースの場所によって同じ値になるとは限りません。
- またコースが設営された会場が違うと、コース上の同じ場所でもカラーセンサの取得する値は違う可能性があります。



例えば、照明にスポットライトがあると、場所によって明るさは均一ではなくなります。

超音波センサ

- 前方障害物との距離を
  計測するためのセンサです。
  - 片方から超音波を発射し、
    障害物から跳ね返った超音波を
    もう片方で検知します。
  - 発射から検知までの時間と、
    音速を利用して測定しています。



- 注意点
  - 他に超音波を発生する装置が近くにあると誤検知します。
  - 超音波は反響するため、
    測距の間隔が短いと誤検知することがあります。
  - 検知するものの形状によっては超音波が
    反射しなかったり、別の方向へ反射したりし、
    測距できないことがあります。

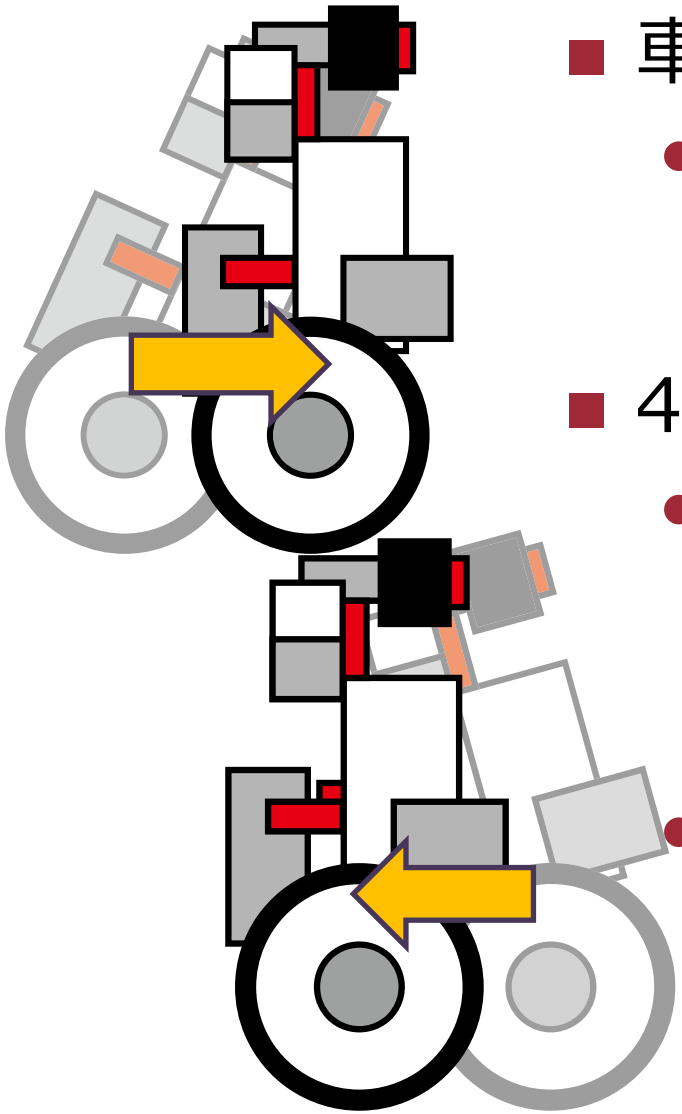
モータ

- 回すスピードをPWM（パルス幅変調）で変えられます。
  - ある一定時間の、何パーセントの時間にモータを回すかという制御
  - バッテリ残量にかかわらず一定速度で回すには？
- トルク
  - 速く回すと高トルク、遅く回すと低トルク
    - 段差を上りきるには？
- エンコーダ
  - モータの回転角度を1度単位で取得できます。
  - エンコーダは出力軸ではなく内部モータ側にあります。
    - つまりギアによる「遊び」の回転を検知することができません。

倒立振子ライブラリ

プライマリークラスでは2輪倒立型走行体を使います
倒立させるためのライブラリの使い方を学びます

倒立振子ライブラリ



- 車体の倒立を制御するライブラリ
  - 前に倒れそうになったら前進、
    後ろに倒れそうになったら後進を
    短時間に繰り返して倒立します。
- 4msごとに呼ぶ
  - ジャイロセンサの値取得可能な
    待ち時間の制限で4msごとに
    呼ばれることを前提に
    設計されています。
  - RTOSの力を借りると楽かも！

使い方の注意点

- 関数呼び出し（サンプルソースを参考に）
  - 初期化：balance_init
  - センサ値からモータパワー計算：balance_control

- ジャイロセンサーのオフセットを0に合わせるために、走行体を停止した状態でセンサをリセットすること

EV3の開発環境

EV3の開発環境の、種類と特徴について学びます

利用可能な開発環境

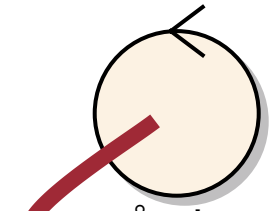
名称	EV3RT	MonoBrick	leJOS
OS種別	TOPPERS/HRP2 (RTOS)	Linux	
言語	C、C++、mruby	C#	Java
統合開発環境	–	Mono-Develop	Eclipse
デバッガ	–	○	○
プログラム転送	USBケーブル Bluetooth SDカード	USBケーブル Bluetooth Wi-Fi SDカード	USBケーブル Bluetooth Wi-Fi SDカード

- EV3RT β6より、USBケーブルを使ったプログラム転送が可能になっています
- 統合開発環境を使うと、書いたソースコードを登録するだけでコンパイルできて便利です
- デバッガを使うと、動作中のプログラムを任意の場所で止めて、変数確認ができるので便利です
- アドバンストクラスでは他にも「ev3dev(Python他)」が利用可能です。

EV3開発環境構築ガイド　https://github.com/ETrobocon/etroboEV3/wiki

OSによる制御方式の違い

- 装置を駆動する方式は主に2種類に大別される

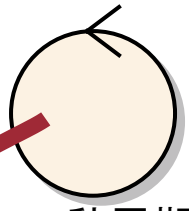


プロセス

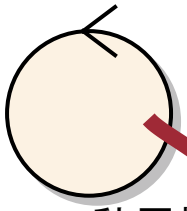
非RTOSでイベント駆動する
コード記述例

```
main()
{
   bool stopFlag=false;
   while( stopFlag )
   {
      int eventType=GetEvent();
      switch( eventType )
      {
         case DEVICE:
            …処理…
            break;
         case STOP
            stopFlag=true;
      }
   }
}
```

Sleep(x)などで時間稼ぎを入れて定期周期を作ることも多い



4m秒周期
タスク



100m秒周期
タスク

RTOSで定期周期する
コード記述例

```
TASK(task4ms)
{
   ScanSensor();
   Balance();
   ActuateMotor();
}

TASK(task100ms)
{
   ScanButton();
   PowerControl();
}
```

定期周期で実行するが
何も仕事がないこともある

RTOS (EV3RT)

Real Time Operating System

代表的な仕組み

- RTOS とは
  - 特徴：リアルタイム応答性能
    - **一定時間内**に、指定した**動作を実行**
    - 倒立振子制御など、**時間制約**の厳しい処理に適している
  - アプリケーションプログラムの実行
    - アプリケーションが **OS と一体**になるものも
    - アプリケーションを一つしか動かせない場合も
  - 汎用 OS (Linux, Windows, OS X, etc.) との比較
    - 動作の「**軽さ**」（**起動時間**、**応答時間**）：　◎
    - マルチプロセス、**動的ロード**、仮想記憶の有無：△
    - **メモリ保護**、**カーネル（OS 本体）保護**の有無：△

タスク（スレッド）切り替えや割り込み応答の時間をなるべく**短く**するために、汎用 OS に比べると、**機能に制限**があります。

汎用 OS に比べると制限がある反面、低機能なプロセッサでも十分動く！

代表的な仕組み

- リアルタイム制御
  - 実現要素
    - **タスク**と割り込み通知
      - タスク：汎用 OS のスレッドに相当
    - 定周期動作：**一定の時間間隔**で動作を実行

ハードウェアによる**割り込み**を OS が応答処理。OS が、必要に応じてアプリケーションの**タスク**に通知（アラームやイベントフラグの発火）。

汎用 OS とは異なり、RTOS では、このタイミングが極めて**正確**です。

RTOS でのプログラミング：タスクが実行する処理を、関数として実装

複数のタスクが存在する場合、最も優先度の高いタスクに CPU が割り当てられて実行され、そのタスクが終了したり待ち状態になるまで、それより低い優先度のタスクは実行されません：**絶対優先度**（RTOS の特徴）

  - 実現方策（定周期動作）
    - ループ内でのスリープ
      - 一定の時間間隔になるようスリープ関数で調整

複数の定周期動作を行う場合（例：倒立振子制御と Bluetooth 通信）、一つのタスクでスリープ関数を使うだけでは、**正確なタイミング合わせ**が難しい。

RTOS は、タスク起動のオーバーヘッドが小さいので、数[ms]おきでも問題ない。

複数のタスクを動かす時の方策

    - 周期起動タスク・周期ハンドラ
      - 一定周期で**繰り返し起動（実行）**されるタスク
    - 周期アラームによる**起床** + ループ中での**休止**

スリープ関数なしでも大丈夫。

EV3RT

- EV3RT の特徴
  - カーネル**保護**／メモリ**保護**
  - アプリケーションの**動的ローディング**
  - 実行開始点（エントリポイント）

汎用 OS では普通

一般的な C/C++ プログラムとは違い、main() ルーチンではなく、（メインタスクの）タスク関数から実行が始まります。

より正確に言うと、コンフィグレーションファイル（.cfg）で設定されたタスクのうち、起動状態での生成を指定されたものの中で、最高優先度のタスクが最初に実行されます。EV3RT では、API 内部の初期化タスクが最初に実行されます。サンプルコードでは、**main_task**() を**メインタスク**のタスク関数として設定しています（初期化タスクより**低**優先度）。

EV3RT での開発

- 汎用 OS との違い
  - タスク（スレッド）を**動的に生成できない**
  - 他のカーネルオブジェクトも同様（アラームやイベントフラグ）

OS の動作を「軽く」するための制限。

静的 API で生成。

全て、アプリケーション（や OS）の**初期化時**にしか生成できない！

- コンフィグレーションファイル（.cfg）
  - 静的 API の呼び出しを .cfg ファイルに記述して設定
  - .cfg ファイルをコンフィグレータで処理

コンフィグレータが、.cfg の内容に従って初期化処理の**コードを生成**する。

- アプリケーションの作成に必要なもの
  - ソースファイルとヘッダファイル（.c, .cpp, .h, .hpp）
  - コンフィグレーションファイル（.cfg）
  - makefile（Makefile.inc）

make コマンドを使ってビルドします。

EV3RT での開発

- C++ソースの書き方
  - main_task(intptr_t) を実装：メインタスク
- タスクの書き方
  - app.cfg にタスク設定を追加
  - ソースファイルにタスク関数を実装

EV3RT での開発

- 注意点
  - 昨年から開発環境を引き継ぐ場合は以下に注意
    - デフォルトのソース置き場が hrp2/sdk/workspace 配下に変更
    - Makefile.app, Makefile.appmodのシンボリックリンクは不要
    - makeのパラメータ変更
      - 動的ローディング形式：make app=<アプリのディレクトリ名>
      - スタンドアローン形式：make img=<アプリのディレクトリ名>
    - Perlをv5.18以上にする必要あり
      - Windows環境ではCygwinを最新にしましょう
  - EV3RTについて判明している問題と対処：
    - https://github.com/ETrobocon/etroboEV3/wiki/problem_and_coping#ev3rt

EV3のAPI

- モータ
- カラーセンサ
- タッチセンサ
- ジャイロセンサ

※APIの詳細や、その他のAPIは下記をご覧ください

- EV3
  - EV3RT C API Reference
    http://www.toppers.jp/ev3pf/EV3RT_C_API_Reference/
  - EV3RT C++ API Reference
    http://www.toppers.jp/ev3pf/EV3RT_CXX_API_Reference/

モータを回す

- ポートAに接続されたモータをフルパワーで回す
  - ev3_motor_set_power (EV3_PORT_A, 100);
    - 第一引数はポート番号、第2引数はパワー
- ポートAに接続されたモータを止める
  - ev3_motor_stop (EV3_PORT_A, *true*);
    - 第2引数は*true*(ブレーキモード), *false*(フロートモード)

モータの回転角を取得する

- Aポートのモータ回転角をリセットする
  - ev3_motor_reset_counts (EV3_PORT_A);

- Aポートのモータ回転角を取得する
  - int32_t counts =
    ev3_motor_get_counts (EV3_PORT_A);

カラーセンサで明度を取得する

- ポート2のカラーセンサを初期化
  - ev3_sensor_config(EV3_PORT_2, COLOR_SENSOR);
    - ポート2がカラーセンサであることを設定

- ポート2のカラーセンサから明度を取得
  - uint8_t light =
    ev3_color_sensor_get_reflect(EV3_PORT_2);

タッチセンサ

- ポート1のタッチセンサの状態を検出する
  - bool_t isPressed =
    ev3_touch_sensor_is_pressed(EV3_PORT_1);

ジャイロセンサ

- ポート1のジャイロセンサの角位置をリセットする
  - ev3_gyro_sensor_reset(EV3_PORT_1);

- ポート1のジャイロセンサで角速度を測定する
  - int16_t rate =
    ev3_gyro_sensor_get_rate(EV3_PORT_1);

競技上のお願い

- 自動停止機能
  - 走行体回収が簡単になるように動きを止めてください
- 改変するときの手続き
  - 提供されたライブラリを改変して使うときは、
    みんなも一緒に使えるようにしてください
- 外乱の影響
  - 「コースに魔物が住んでいる」と言われますが
    完走目指して競技会場の外乱に注意してください

自動停止機能

- 難所では走行体転倒の可能性があります
- 競技を円滑に進めるために

走行体が転倒などの制御不能状態になった場合は、
自動的に両輪モータを停止する機能

の搭載を検討してください

- 走行体の転倒状態を検出することは、難所攻略に必要な要素技術の一つでもあります

ライブラリ改変に関して

- まずETロボコン実行委員会が考える競争領域、非競争領域は以下の通りです
  - 競争領域：所定のプラットフォーム環境の上で、目的とする制御(ライントレースなど)を実現するアプリケーションプログラムを、モデリングを使って、いかに分析・設計・実装するか
  - 非競争領域：プラットフォーム・環境
- よって各参加チームは、**ETロボコン技術委員会が準備した環境を使用することを前提**としていますが、それ以外のプラットフォーム・環境の使用、または改変して使用することで走行性能が改善される、あるいは参加者になじみ深い環境なので開発効率が上がる、などの効果が見込まれる場合は、**以下の条件のもとで、改変・新規開発環境等の使用を認めています**
  - 全ての参加者が容易に利用可能であること
  - 参加者全員にその内容や利用手順が公開されていること
  - 7月末日までに、競技規約に基づく申請が行われていること
- 先の条件の最初にある**「全ての参加者が容易に利用可能であること。」**とは、単なるソースコードの公開だけではなく、**他の参加者が利用する場合の技術資料の公開も含んでいます**
- ライブラリ改変の申請をする皆さんにはお手数をおかけすることになりますが、上記の趣旨をご理解の上、ご協力をお願いします