平成２３年度
プロジェクト型実践演習
第2回ハードウェア説明、ソフトウェア説明

平成23年10月8日

主催：福岡市

実施：NPO法人 九州組込みソフトウェアコンソーシアム
（QUEST）

サマリ

- 1. ハードウェア説明
- 2. ソフトウェアの設計
- 3. 開発環境
- 4. 実行手順
- 5. その他

1．ハードウェア説明

- NXTのスペック
- NXTのブロック図
- NXTのハードウェア機能

NXTのスペック

- メインプロセッサー
  - Atmel® 32-bit ARM® processor, AT91SAM7S256
    - – 256 KB FLASH
    - – 64 KB RAM
    - – 48 MHz
- コ・プロセッサー
  - Atmel® 8-bit AVR processor, ATmega48
    - – 4 KB FLASH
    - – 512 Byte RAM
    - – 8 MHz
- ブルートゥース無線通信
  - CSR BlueCoreTM 4 v2.0 +EDR System
  - – Supporting the Serial Port Profile (SPP)
  - – Internal 47 KByte RAM
  - – External 8 MBit FLASH
  - – 26 MHz
- USB 2.0 通信
  - Full speed port (12 Mbit/s)
- 4つの入力ポート
  - 6線インタフェースで、デジタルとアナログの2つをサポート
    - – 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
- 3つの出力ポート
  - 6線インタフェースでエンコーダからの入力をサポート
- ディスプレイ
  - 100 x 64 pixel LCD black & white graphical display
    - – View area: 26 X 40.6 mm
- ラウドスピーカー
  - 8ビット分解能サウンド出力チャンネル
    - – Supporting a sample rate of 2-16 KHz
- 4つのボタンユーザインタフェース
  - ゴム製ボタン
- 電源
  - 6本単三バッテリー
    - – アルカリバッテリー推奨
    - – 充電式リチウムイオンバッテリー1400mAHが利用可能
- コネクター
  - 6線式工業標準コネクターであるRJ12、右サイドアジャストメント

NXTのブロック図

- 公開されている、「LEGO MINDSTORMS NXT Hardware Developer Kit.pdf」より抜粋

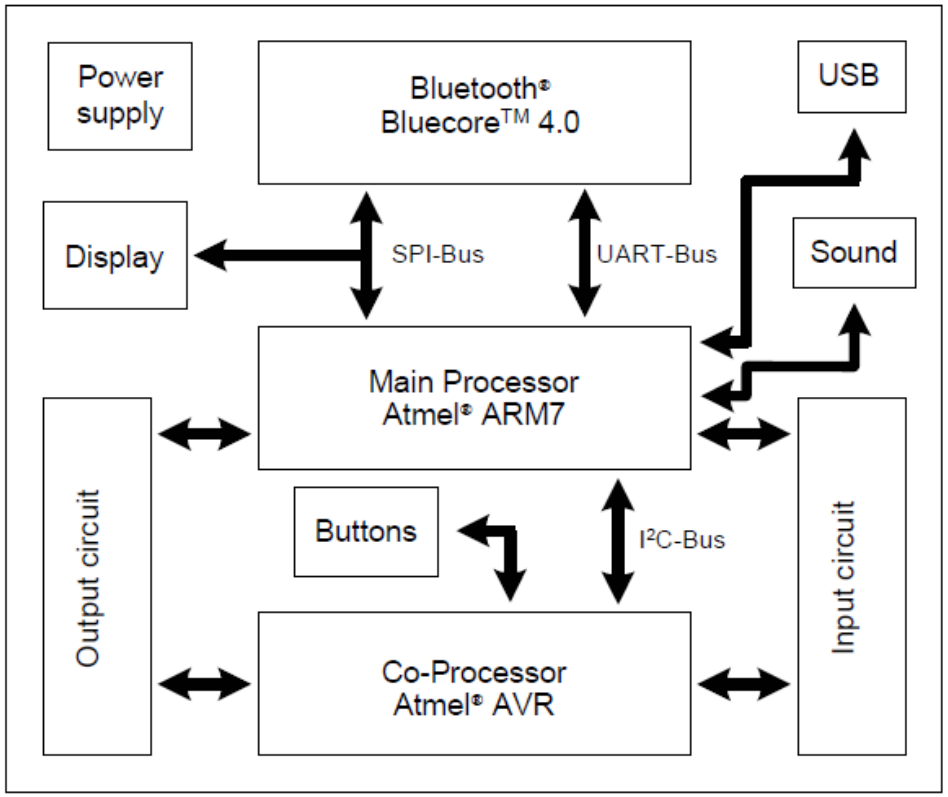
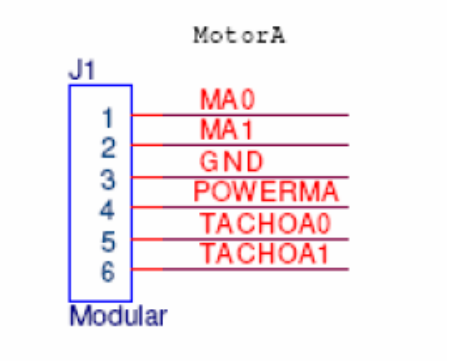


Figure 1: Hardware block diagram of the NXT brick

NXTのハードウェア機能

- 出力ポート
- 入力ポート
- ディスプレイ
- ブルートゥース

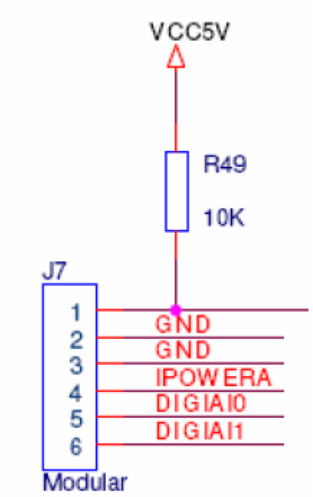
出力ポート



Pin 1, MA0	PWM output signal for the actuators
Pin 2, MA1	PWM output signal for the actuators
Pin 3, GND	Ground signal related to the output supply
Pin 4, POWERMA	4.3 Volt output supply
Pin 5, TACHOA0	Input value that includes Schmitt trigger functionality
Pin 6, TACHOA1	Input value that includes Schmitt trigger functionality

- 出力ポートは、合計3ポート
- モータ駆動用の用途が多い
  - MA0　:駆動装置のためのPWM出力信号
    - モータの片側へ接続
  - MA1　:駆動装置のためのPWM出力信号
    - モータの片側へ接続
  - GND　:接地
  - POWERMA:　4．3V電源供給
  - TACHOA0:シュミットトリガー機能を含む入力値
    - ノイズに強い2値を得られる(波形がパタパタしない)
  - TACHOA1:シュミットトリガー機能を含む入力値
    - ノイズに強い2値を得られる(波形がパタパタしない)

入力ポート



- Pin 1, ANA — Analog input and possible current output signal
- Pin 2, GND — Ground signal
- Pin 3, GND — Ground signal
- Pin 4, IPOWERA — 4.3 Volt output supply
- Pin 5, DIGIAI0 — Digital I/O pin connected to the ARM7 processor
- Pin 6, DIGIAI1 — Digital I/O pin connected to the ARM7 processor

- 入力ポートは、合計4ポート
- センサー系の入力が多い
  - ANA：アナログ入力。10ビットのA/D変換を行う。AVRプロセッサーへ接続
  - GND：接地
  - IPOWERA：4．3V電源供給
  - DIGIAI0、1：デジタル入力。ARM7プロセサーへ接続

ディスプレイ

The pixels within the display are allocated as follows:

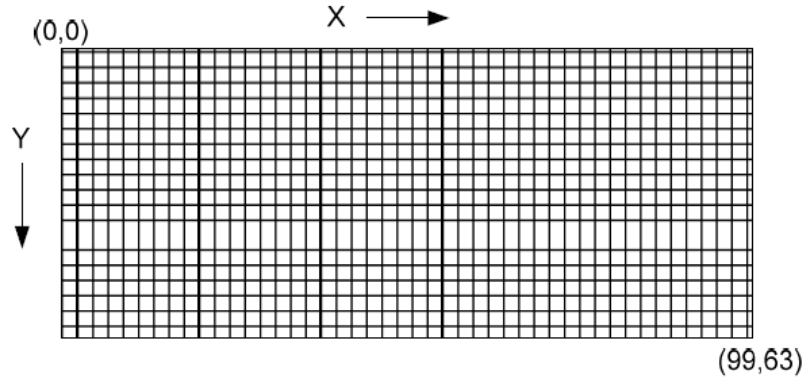


Figure 5: Bitmapping within the display

- 100x64ドット
- STNモノクロディスプレイ
- 英数字記号を表示できる
- 日本語はROM容量の関係上事実上無理

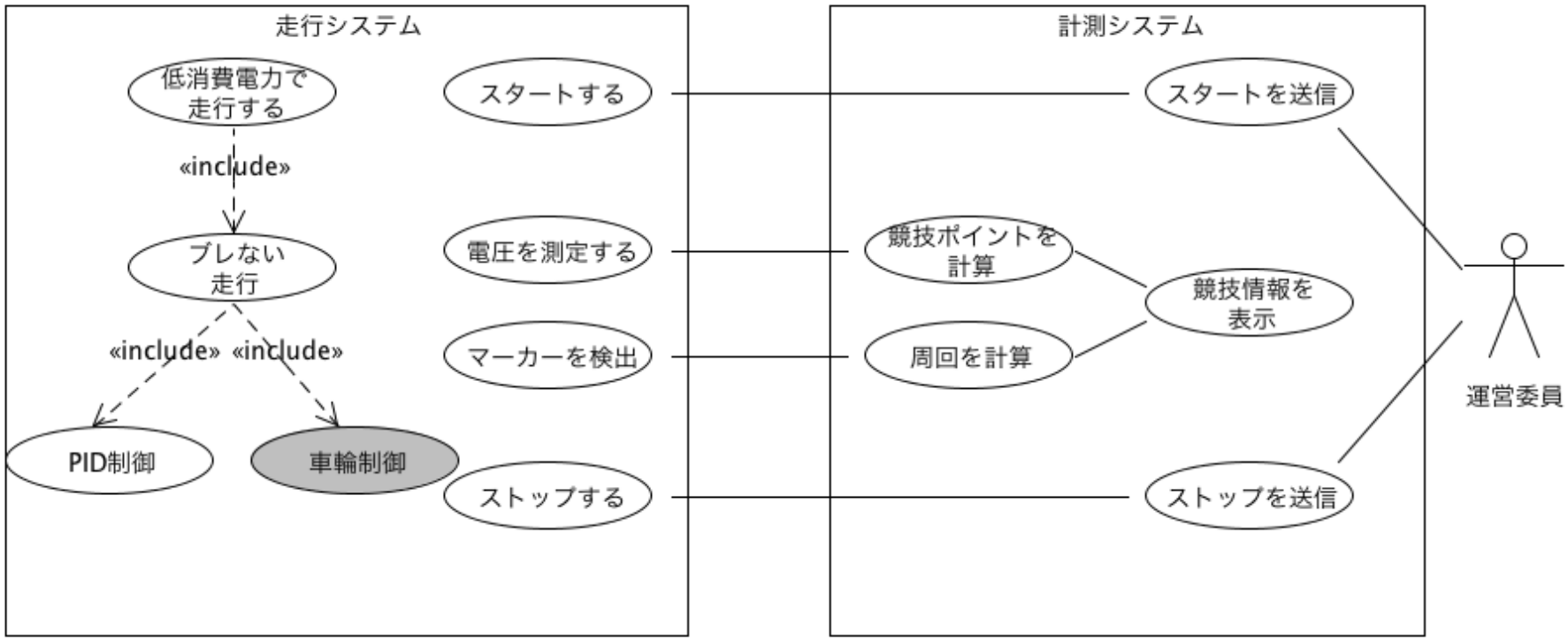
ブルートゥース

- SPP（Serial Port Profile）をサポートしている
- パソコンやNXTとSPPを通じて通信可能
- 到達可能距離は、約10m
- マスター/スレーブモードで通信可能
  - GETC2011の場合、PCがマスターでNXTがスレーブとなる。

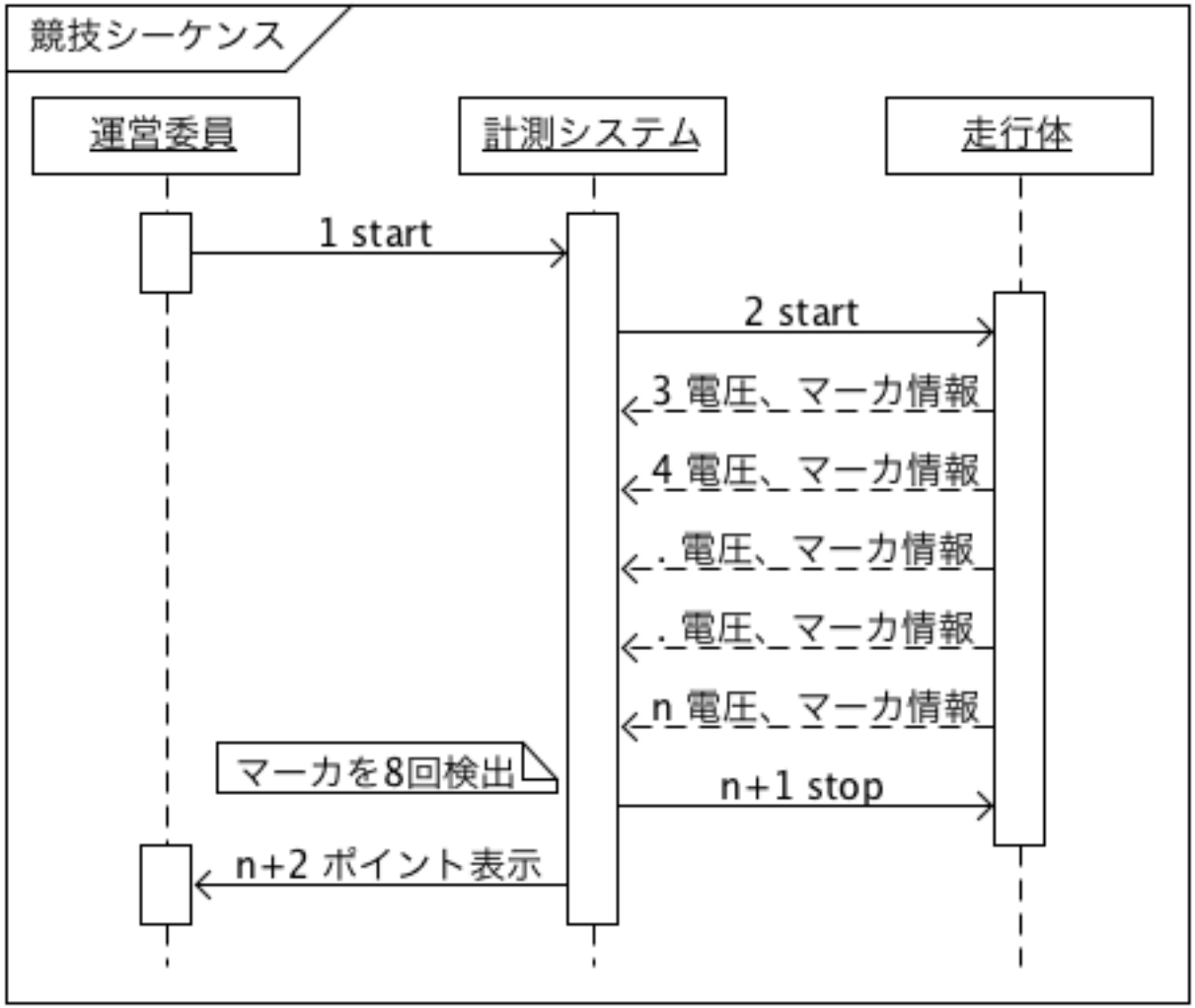
2. ソフトウェア説明

- ユースケース
- シーケンス図
- 要求分析
- クラス図
- 状態遷移図

Green ET Challenge2011のユースケース図

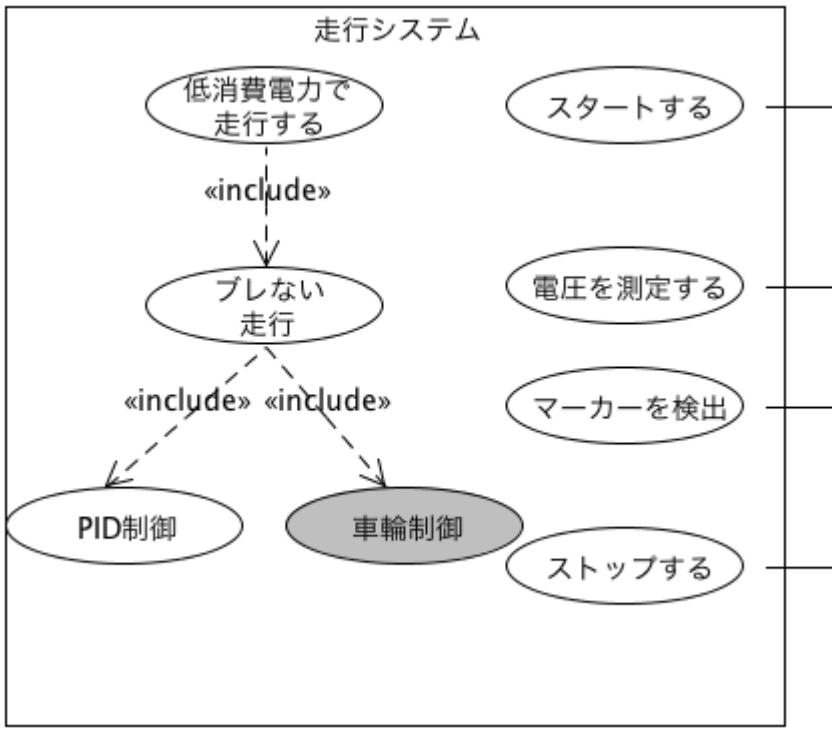


競技シーケンス図


競技シーケンス
運営委員
計測システム
走行体
1 start
2 start
3 電圧、マーカ情報
4 電圧、マーカ情報
. 電圧、マーカ情報
. 電圧、マーカ情報
n 電圧、マーカ情報
マーカを8回検出
n+1 stop
n+2 ポイント表示

要求分析

- 要求
  - スタートする
  - ストップする
  - 電圧を測定する
  - マーカーを検出する
  - 低消費電力で走行する



要求分析

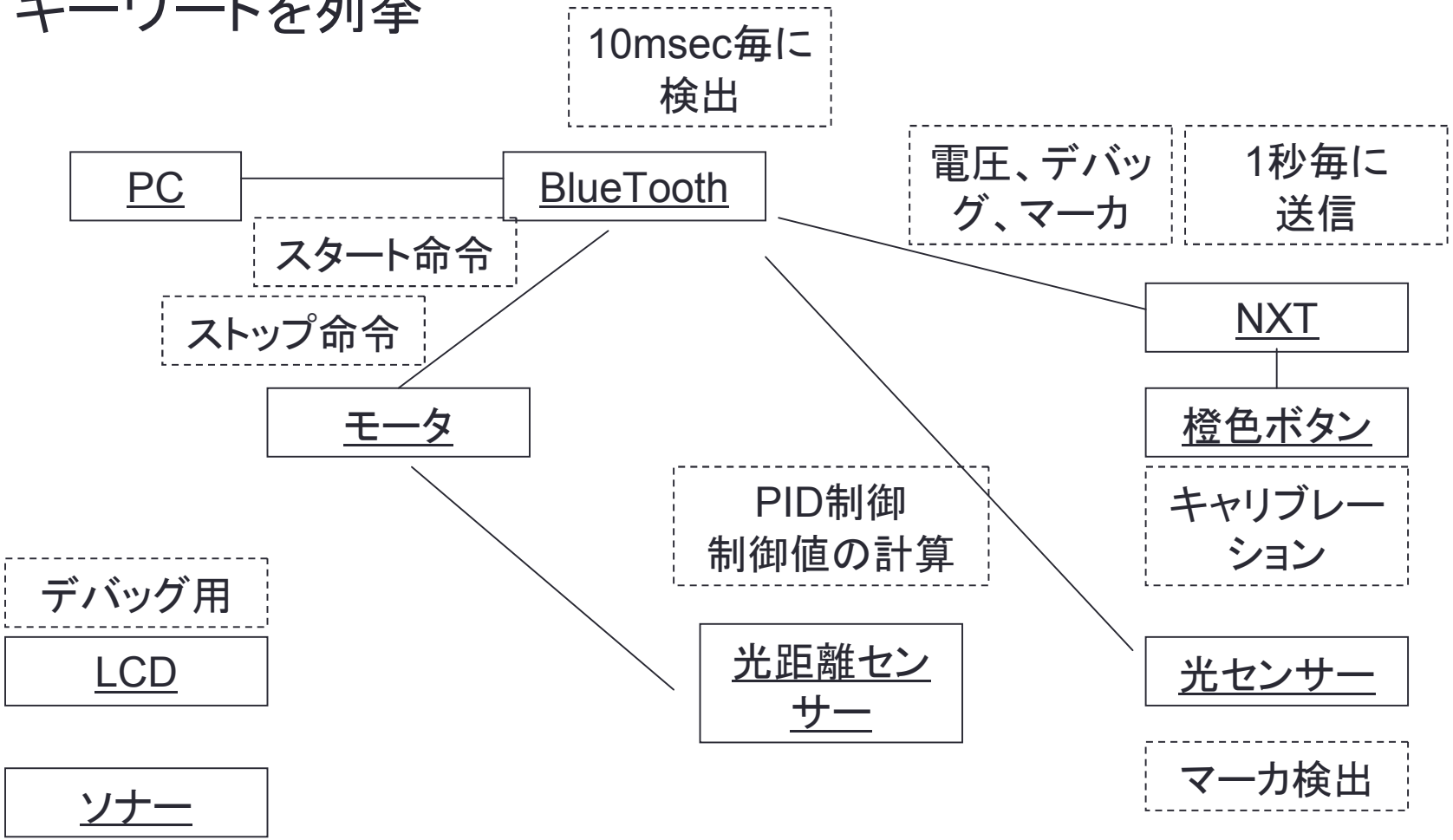
- 実現方法
  - スタート/ストップする
    - PCからのスタート、ストップの命令をBlueTooth無線によって10msec毎に検出しモータへスタート及びストップの命令を実行する
  - 電圧を測定する
    - 10msec毎に電圧を測定し、1秒間の間平均を取りその平均を測定値とする
  - マーカーを検出する
    - 10msec毎に光センサーを検出し、バッファーに貯めて連続して白であれば、PCへ「マーカー有り」と通知する。未検出時は「マーカー無し」を通知する
      - PC側では、黒から白へ変化した時にマーカー検出とする。
    - キャリブレーション用に四角橙色ボタンを利用する
  - 測定値の送信
    - 電圧、マーカ及びデバッグ情報は、1秒単位でPCへBlueTooth無線によって送信する

要求分析

- 実現方法
  - 低消費電力で走行する
    - 振れずに、負荷の少ない走行をさせる
      - 負荷の少ない走行のためには両壁に当たらないように壁の中央で走行させる
      - PID制御を使って、ブレない走行をさせる
      - PID制御の結果得られた補正値を適正な値に変換してモータへ伝達する
        - 適性な値で変換するために、補正値と制御値の関係を明確にする
  - その他
    - デバッグ用にLCDディスプレイを利用する
    - デバッグ用に1秒単位でPCにデバッグデータを送信する
    - ソナーセンサーは他のチームと干渉する恐れがあるので使わない

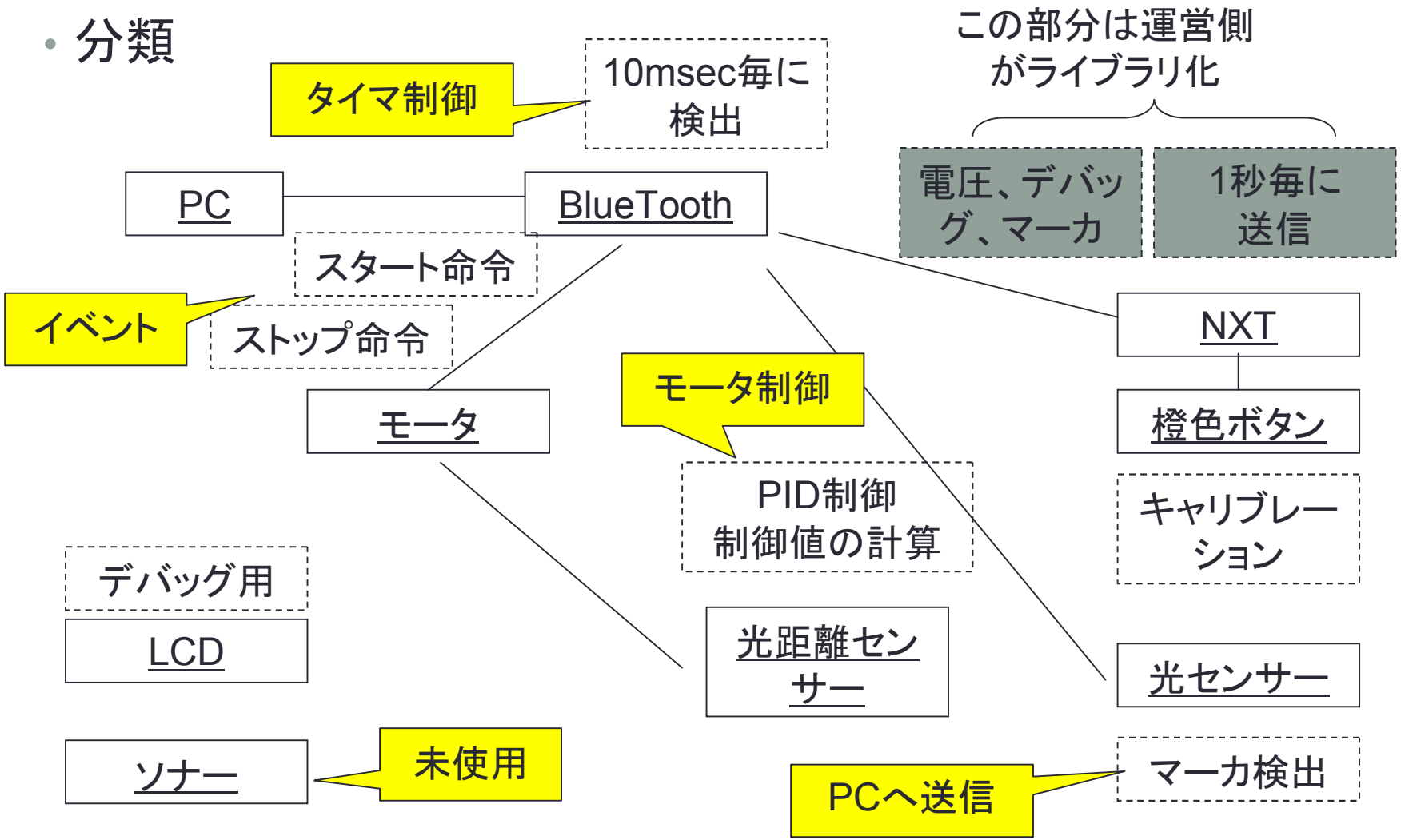
設計

- キーワードを列挙


10msec毎に検出
PC
BlueTooth
電圧、デバッグ、マーカ
1秒毎に送信
スタート命令
ストップ命令
NXT
モータ
橙色ボタン
PID制御
制御値の計算
キャリブレーション
デバッグ用
LCD
光距離センサー
光センサー
ソナー
マーカ検出

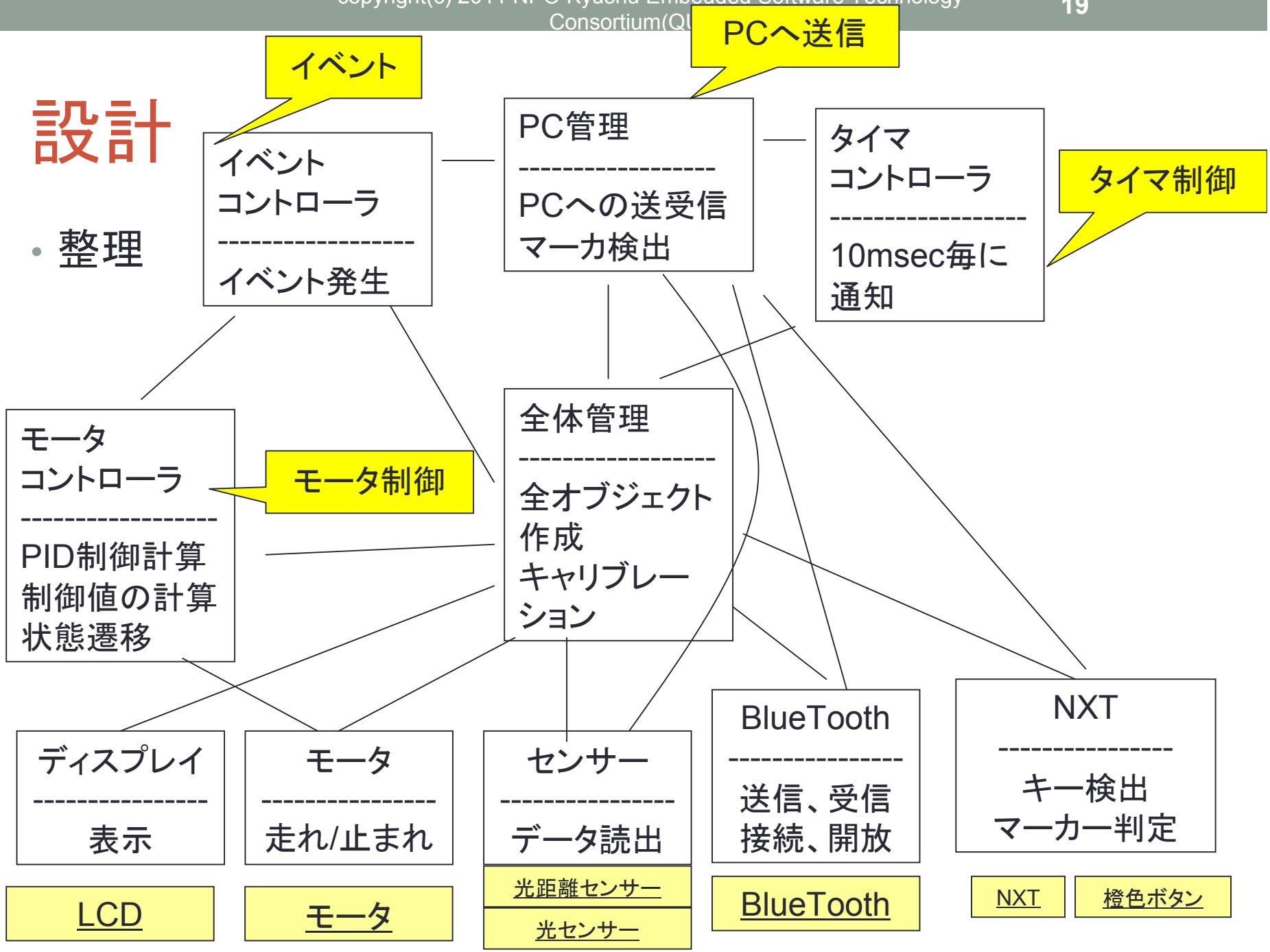
設計

- 分類


この部分は運営側がライブラリ化
タイマ制御
10msec毎に検出
PC
BlueTooth
電圧、デバッグ、マーカ
1秒毎に送信
スタート命令
イベント
ストップ命令
NXT
モータ制御
モータ
橙色ボタン
PID制御
制御値の計算
キャリブレーション
デバッグ用
LCD
光距離センサー
光センサー
ソナー
未使用
PCへ送信
マーカ検出

設計

- 整理


イベント
PCへ送信
イベント
コントローラ

イベント発生
PC管理

PCへの送受信
マーカ検出
タイマ
コントローラ

10msec毎に
通知
タイマ制御
モータ
コントローラ

PID制御計算
制御値の計算
状態遷移
モータ制御
全体管理

全オブジェクト
作成
キャリブレー
ション
ディスプレイ

表示
モータ

走れ/止まれ
センサー

データ読出
BlueTooth

送信、受信
接続、開放
NXT

キー検出
マーカー判定
LCD
モータ
光距離センサー
光センサー
BlueTooth
NXT
橙色ボタン

ところで

- よくある仕様変更
  - イベントが追加、削除された。
    - それによって、通知して欲しいコントロラーが増減した。
  - タイマ割り込み（10msec）毎に通知して欲しいコントローラが増減した
  - タイマ割り込み（100msec）毎に通知して欲しいコントローラが動的に増減する仕様になった
- 困った
  - 登録したオブジェクトに対して、一斉に同報し、しかも同じ関数名を使うことにより、オブジェクトによって違う動作をさせる簡単なメカニズムが欲しい。

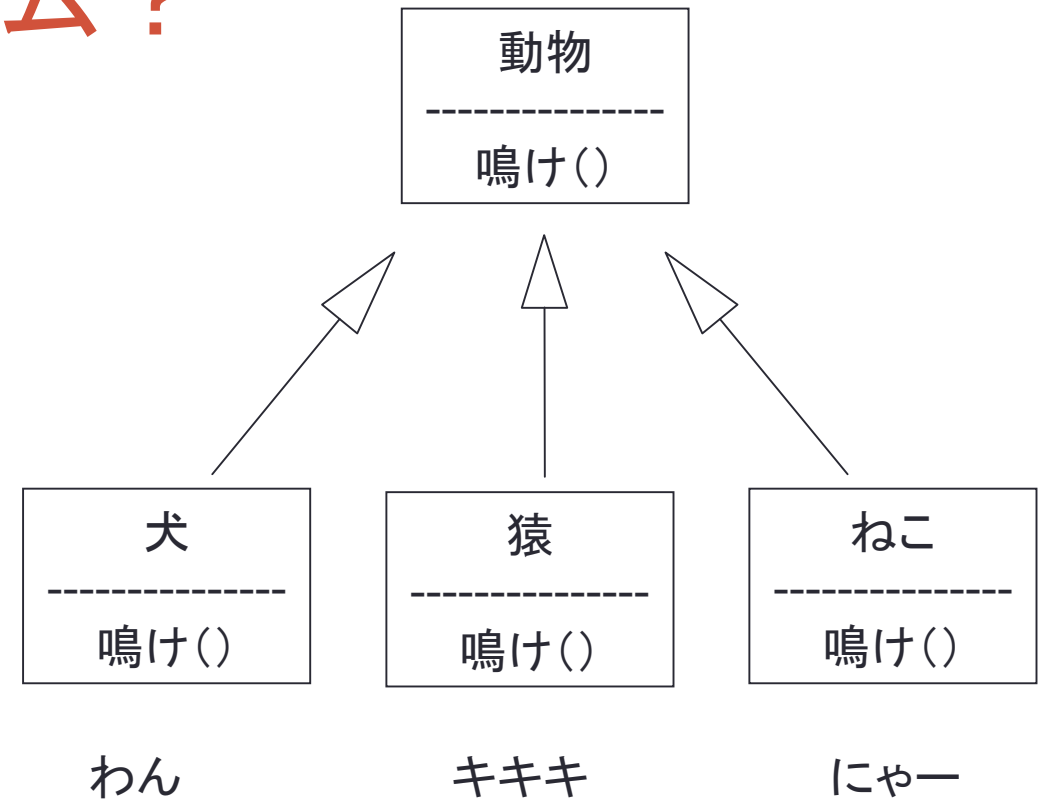
対策

- ポリモーフィズム（多様性、多態性）による対策
  - 全体の構成を変更することなく、追加、削除のルーチンワークでオブジェクトの追加、削除が実現できる。
  - 同報通知機能が実現できる
  - １種類の同報でなく、数種類の同報が実現できる

ポリモーフィズム？

よくある例
動物　* ani = new 動物();
犬　　* dog = new 犬();
猿　　* monkey = new 猿();
ねこ　* cat = new ねこ();

ani = (動物*)dog;
ani->鳴け()//わん
ani = (動物*)monkey;
ani->鳴け()//キキキ
ani = (動物*)cat;
ani->鳴け()//にゃー



この時に、重要なのはaniという汎用的なオブジェクトで一元的に管理できるということである。
すなわちaniは動物クラスなので配列に入れたり、リスト構造をとったりと自由度が高まる。
dogやcatは同じ配列にそのままだと代入できない。

同報の実現

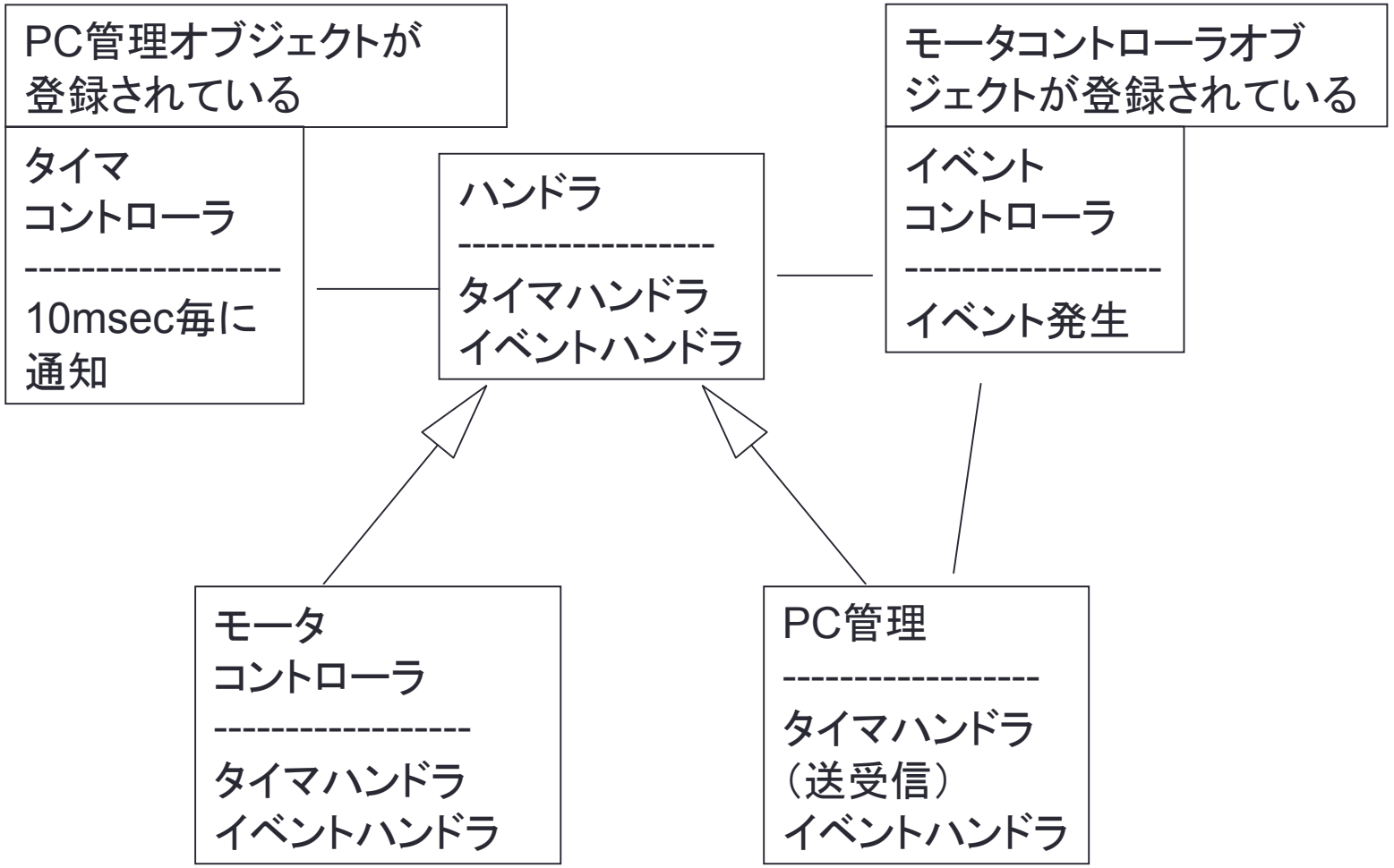
```
動物  *ani [3];
 ani[0] = (動物*)(new 犬());
 ani[1] = (動物*)(new 猿());
// ani[2] = new ねこ(); // これはコンパイルエラーする
 ani[2] = (動物*)(new ねこ());

for (int I = 0 ; I < 3; i++){
        ani[i] -> event();// わん、キキキ、にゃー
}
```

更に、aniを配列ではなくてチェーン構造にすれば、追加、削除も動的に実現できる

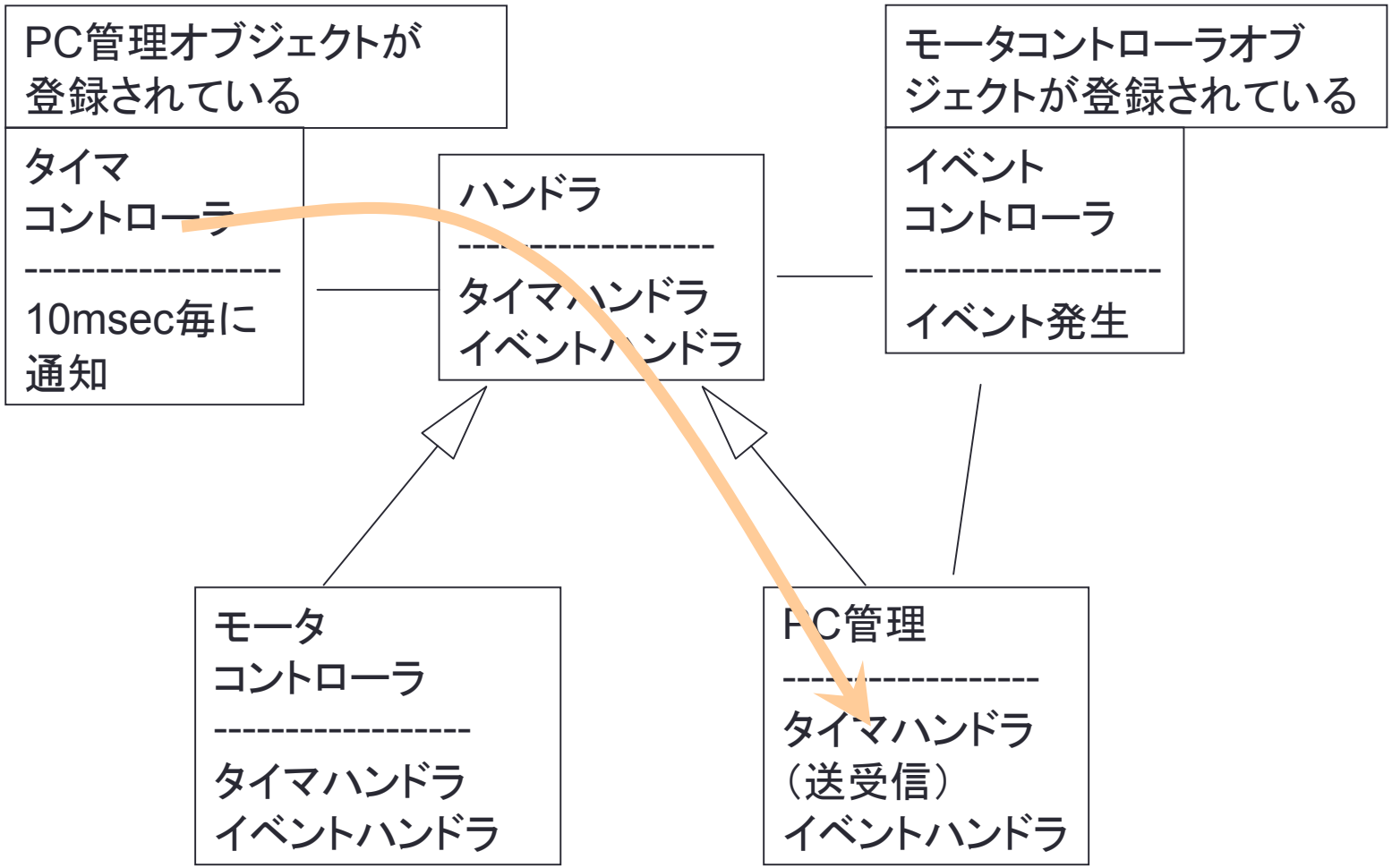
設計

- 整理


PC管理オブジェクトが
登録されている
モータコントローラオブ
ジェクトが登録されている
タイマ
コントローラ
10msec毎に
通知
ハンドラ
タイマハンドラ
イベントハンドラ
イベント
コントローラ
イベント発生
モータ
コントローラ
タイマハンドラ
イベントハンドラ
PC管理
タイマハンドラ
（送受信）
イベントハンドラ

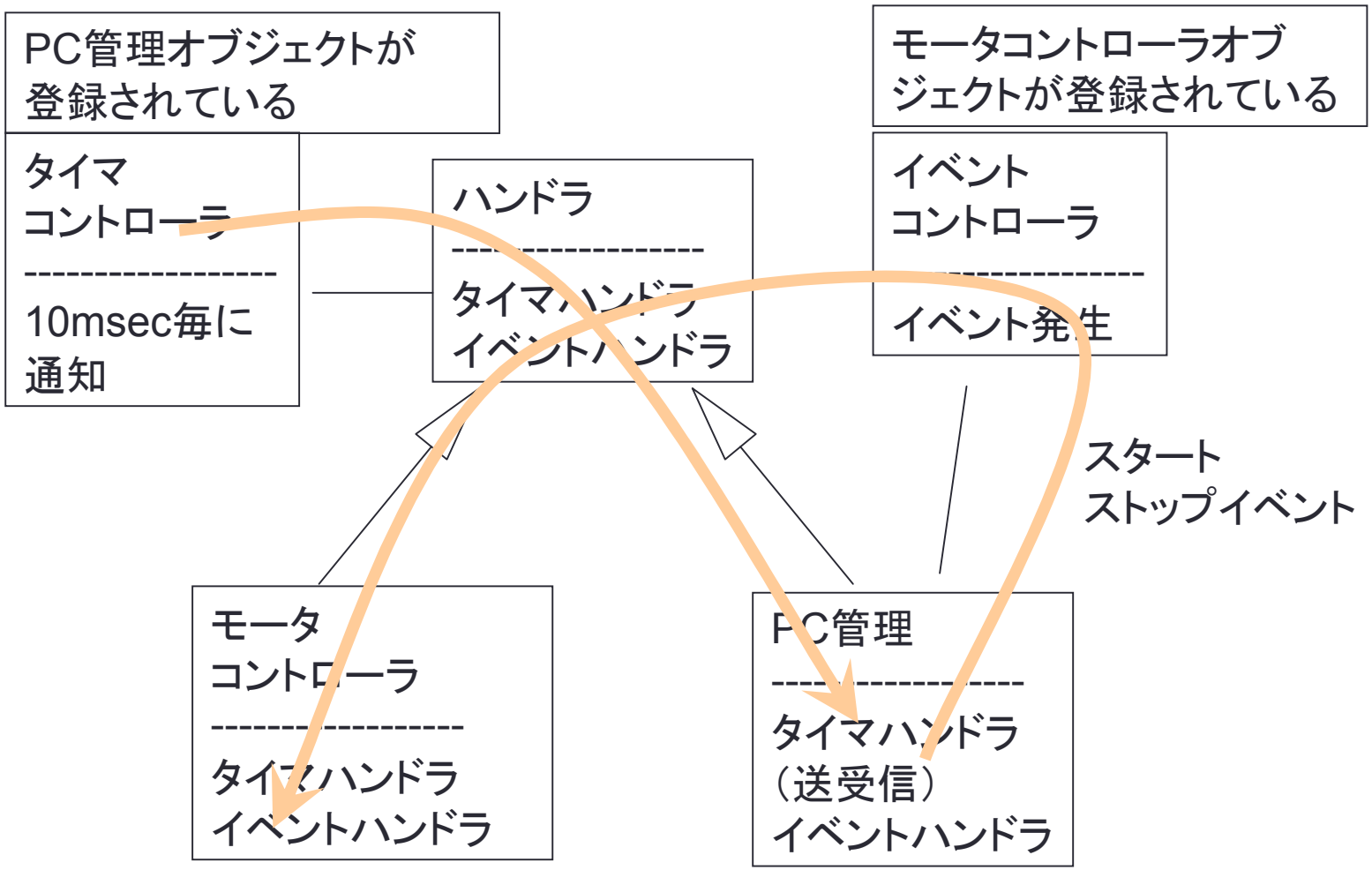
設計段階での検証

- タイマ割り込み(10msec毎)

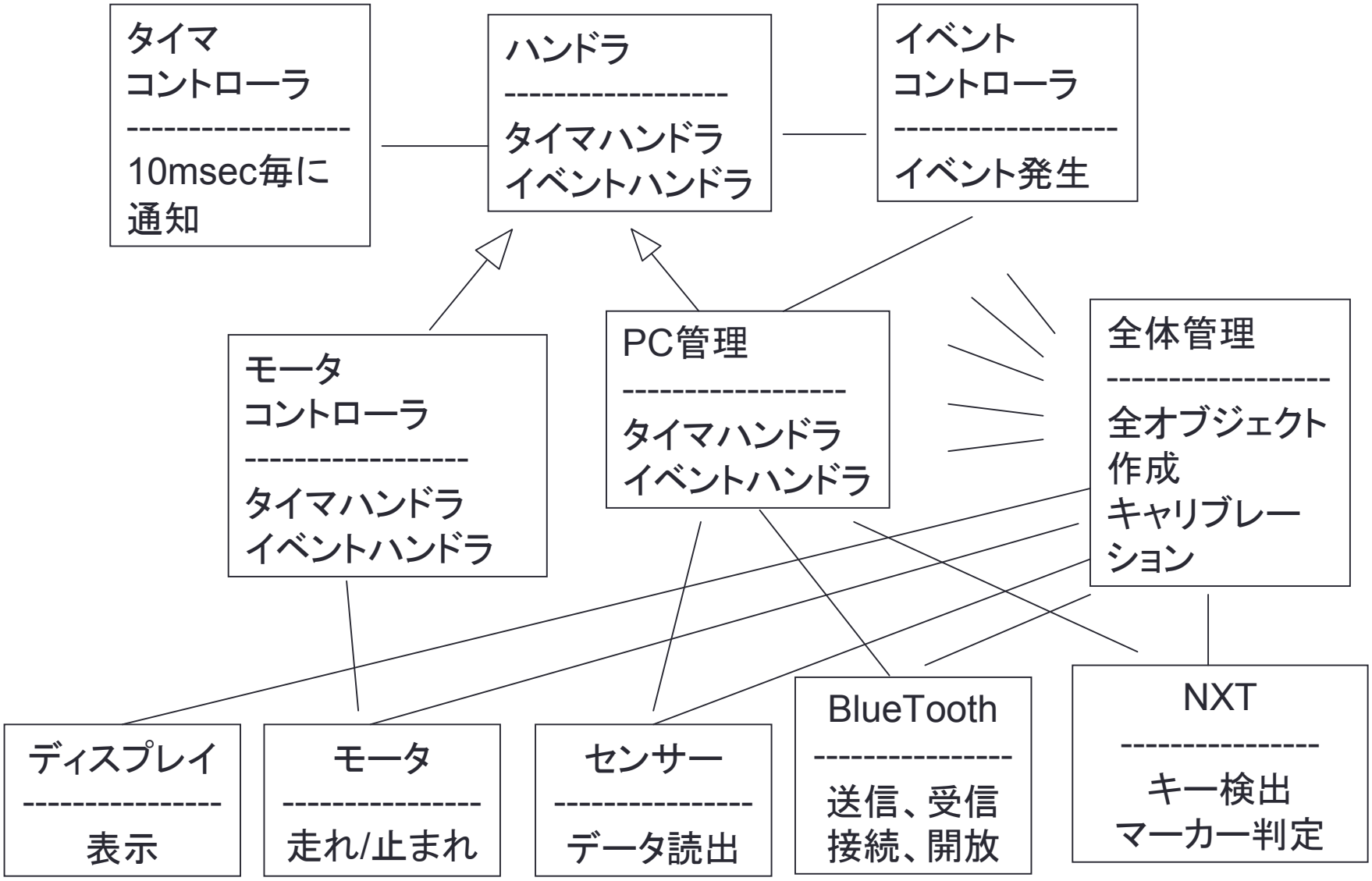


設計段階での検証

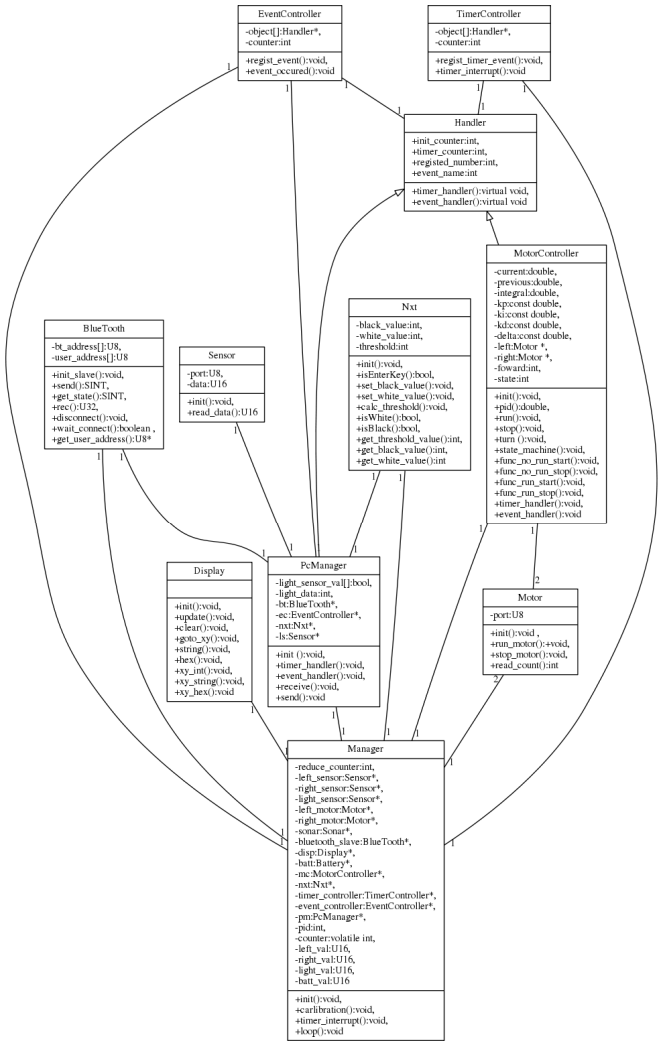
- イベント読み込み(10msec毎)



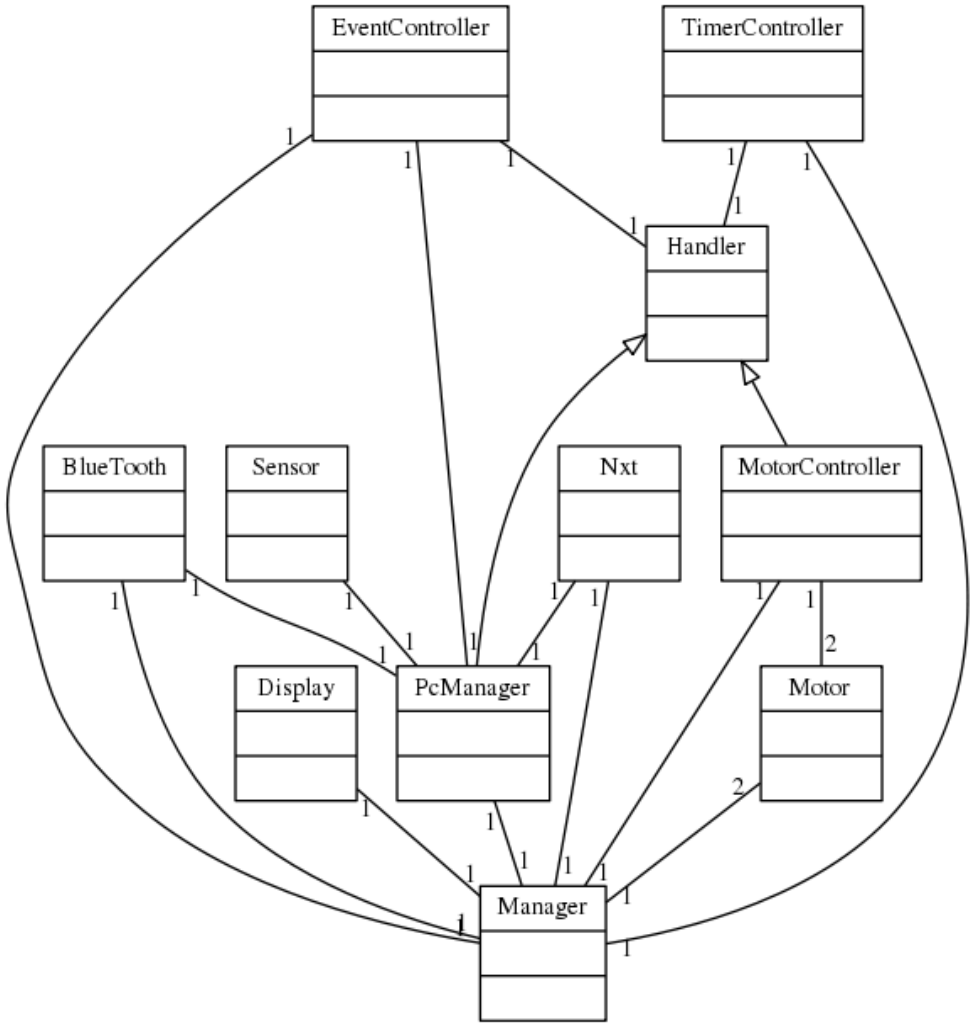
設計(最終)


タイマ
コントローラ
10msec毎に
通知
ハンドラ
タイマハンドラ
イベントハンドラ
イベント
コントローラ
イベント発生
モータ
コントローラ
タイマハンドラ
イベントハンドラ
PC管理
タイマハンドラ
イベントハンドラ
全体管理
全オブジェクト
作成
キャリブレー
ション
ディスプレイ
表示
モータ
走れ/止まれ
センサー
データ読出
BlueTooth
送信、受信
接続、開放
NXT
キー検出
マーカー判定

Demoプログラムのクラス設計



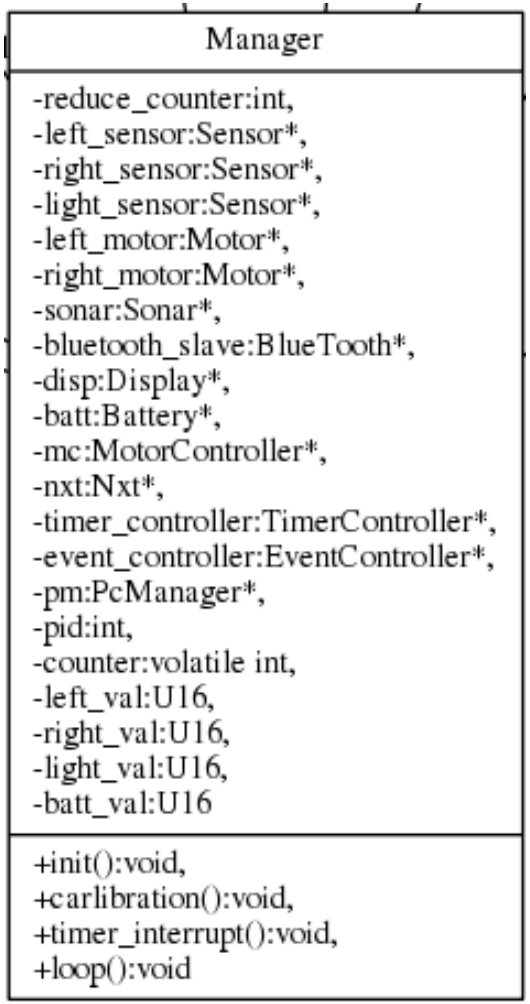
Demoプログラムのクラス設計



Demoプログラムのクラス説明

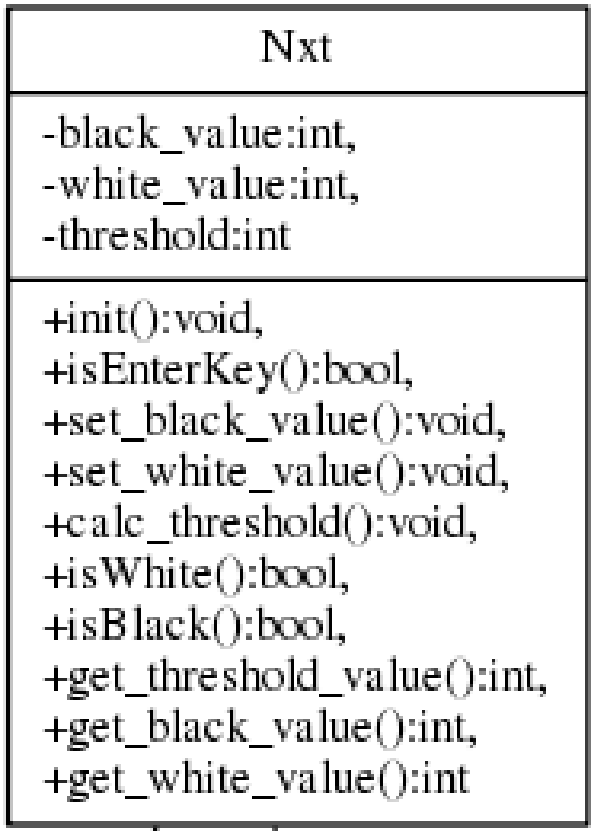
- Manager
- Nxt
- Sensor
- BlueTooth
- Display
- Motor
- TimerController
- EventController
- Handler
- PcManager
- MotorController

Manager



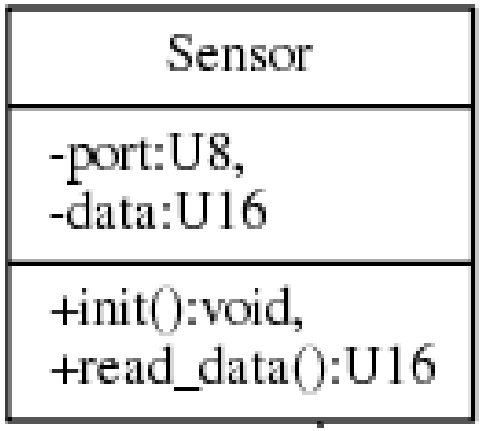
- すべてのクラスのオブジェクトを管理するクラス
- init()
  - オブジェクトの初期化
- carlibration()
  - マーカ検出のためのキャリブレーションを行う
- timer_interrupt()
  - 1msec毎に割り込みフック関数から呼ばれて、TimerControllerへ通知する
- loop()
  - 5msec毎にMotorControllerに対してpid計算及びモータの制御を要求する

Nxt



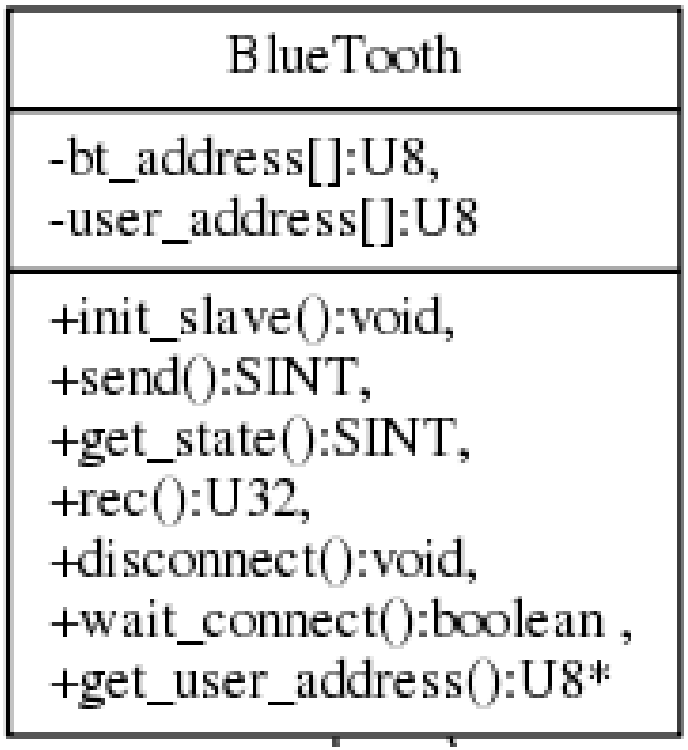
- Nxtのボタン及びマーカ情報を管理するクラス
- init()
  - Nxtの初期化を行う
- isEnterKey()
  - 橙色のキーが押されたかを検出する
- set_black_value()
  - キャリブレーションの結果黒の値をセットする
- set_white_value()
  - キャリブレーションの結果白の値をセットする
- calc_thresold()
  - 白と黒から閾値を計算する
- isWhite()
  - 白であるかを判定する
- isBlack()
  - 黒であるかを判定する
- get_threshold_value()
  - スレッショルド値を取得するためのアクセスサー
- get_black_value()
  - 黒の値を取得するためのアクセスサー
- get_white_value()
  - 白の値を取得するためのアクセスサー

Sensor



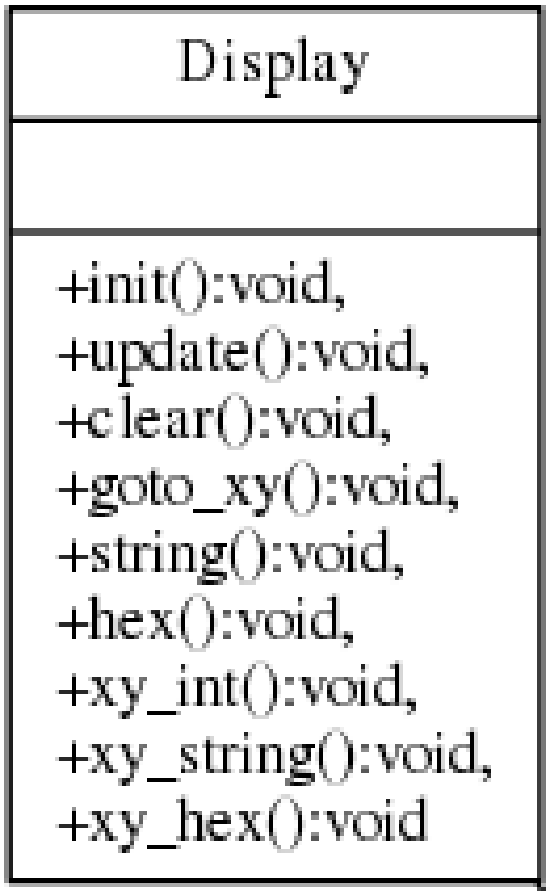
- 次の2つのセンサーを管理するクラス
  - 光距離センサー
  - 光センサー
- オブジェクトは3つ作成される
  - 左用光距離センサー
  - 右用光距離センサー
  - マーカー用光センサー
- init()
  - 初期化
- read_data()
  - データの取得
- 光距離センサー、光センサーとも取得関数APIは同じものが使える。
- APIが違うような他のセンサーを使う場合は汎化したほうが良いかもしれない。

BlueTooth



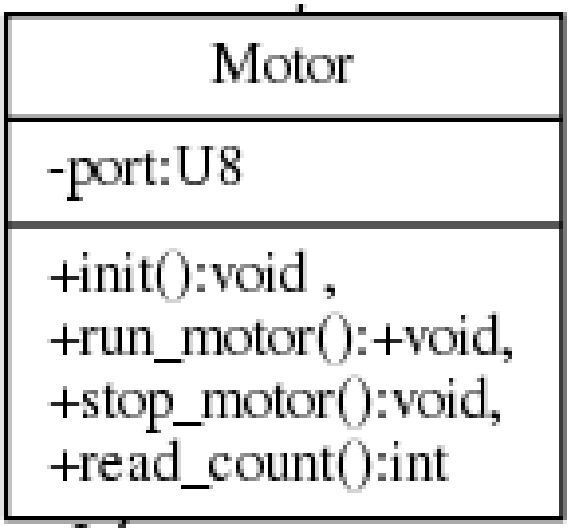
- BlueToothを管理するクラス
- init_slave()
  - スレーブモードで初期化する
- send()
  - データを送信する
- get_state()
  - 現在の通信状態を取得する
- rec()
  - データを受信する
- disconnect()
  - 通信を終了する
- wait_connect()
  - コネクトするまで待つ
- get_user_address()
  - BlueToothのmacアドレス下位2バイトを返却する。(未使用)

Display



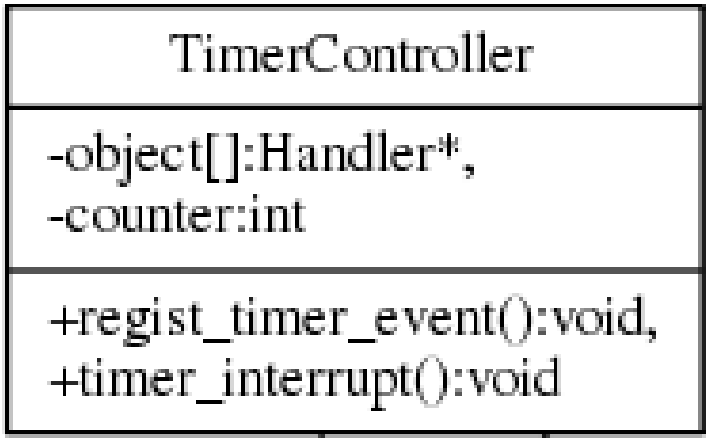
- LCDディスプレイを管理するクラス
- init()
  - 初期化する
- update()
  - 描画したあとに呼ぶと画面に更新される
- clear()
  - 画面を全消去する
- goto_xy()
  - キャラクタ座標を移動する
- string()
  - 文字列を表示する
- hex()
  - 16進数の値を表示する
- xy_int()
  - 10進数の値を指定した座標に表示する
- xy_string()
  - 文字列を指定した座標に表示する
- xy_hex()
  - 16進数の値を指定した座標に表示する

Motor



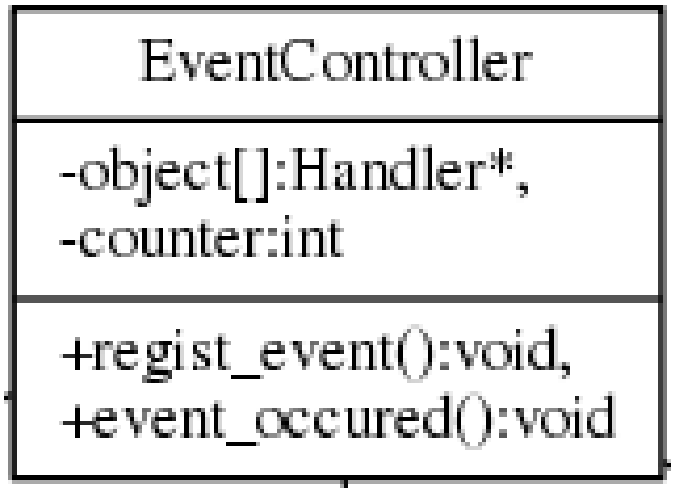
- モータを管理するクラス。Managerクラスによって、左右2つのモータオブジェクトを生成する
- init()
  - モータの初期化。
- run_motor()
  - スピードを与るとそのスピードで回転する
- stop_motor()
  - モータを止める
- read_count()
  - モータのロータリーエンコーダの値を取得する。リセットはinit()で行う。（未使用）

TimerController



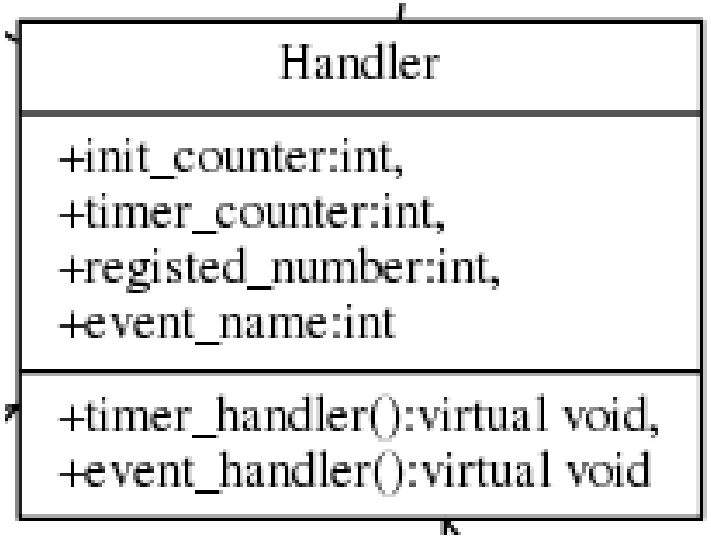
- タイマーを管理するクラス
- 登録されたオブジェクトに対してそのオブジェクトのtimer_handler()関数を呼び出す
- regist_timer_event()
  - Handler*でキャストされたHandlerを継承しているオブジェクトを登録する。10msec以上のタイマも同時にセットする(動的な追加、削除には対応していない)
- timer_interrupt()
  - 登録されたオブジェクトに対して、セットされた時間が来たら、そのオブジェクトのtimer_handler()関数を呼び出す

EventController



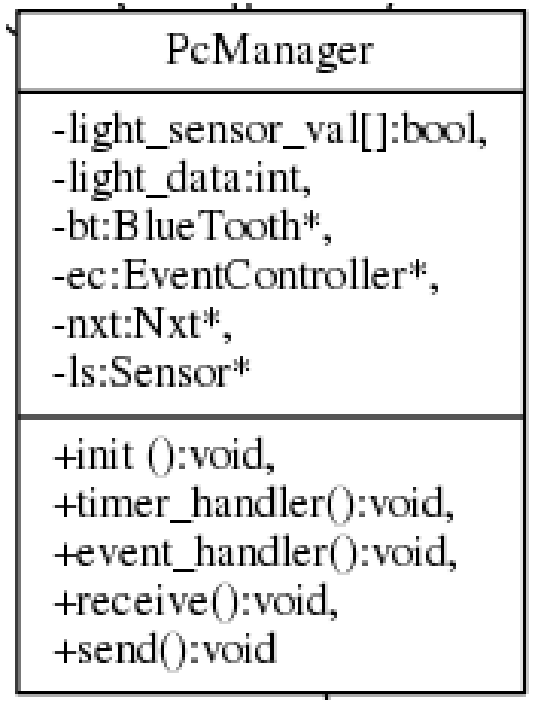
- イベントを管理するクラス
- 登録されたオブジェクトに対してそのオブジェクトのevent_handler()関数を呼び出す
- regist_event()
  - Handler*でキャストされたHandlerを継承しているオブジェクトを登録する。(動的な追加、削除には対応していない)
- event_occured()
  - この関数が呼ばれると、登録されたオブジェクトに対して、そのオブジェクトのevent_handler()関数を呼び出す

Handler



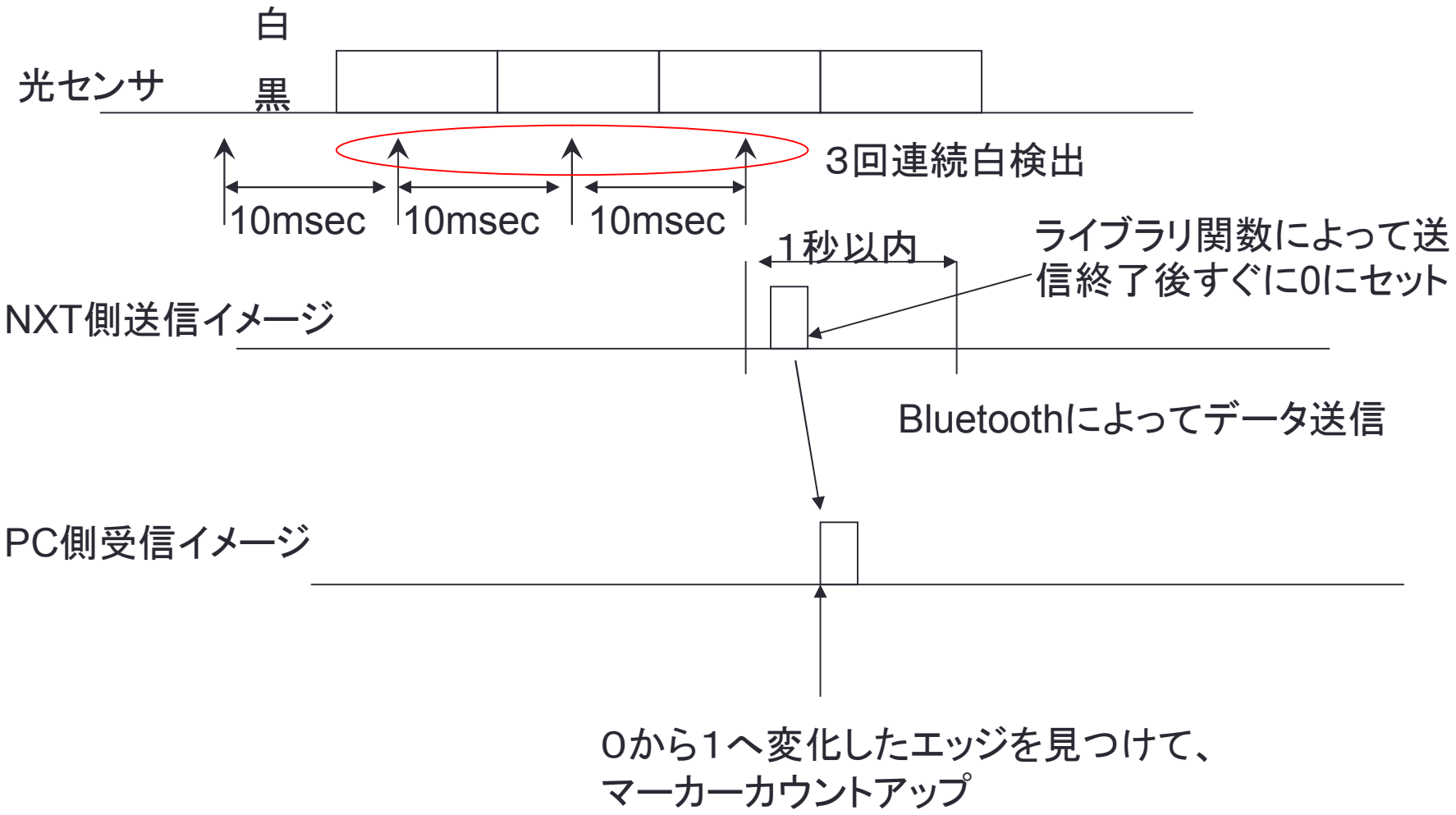
- イベントやタイマーを通知するために使われるクラスであり、通知を受け取るクラスの基底クラスとなる。
- timer_handler()
  - タイマーハンドラーの純粋仮想関数
- event_handler()
  - イベントハンドラーの純粋仮想関数

PcManager

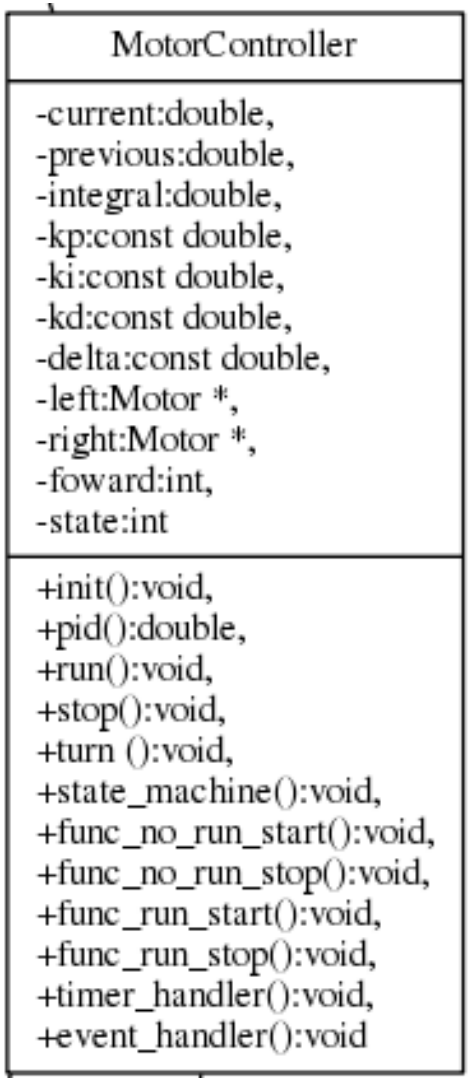


- 計測システム（PC）と走行体との通信を管理するクラス。上位クラスはHandler
- init()
  - 初期化
- timer_handler()
  - 10msec毎に呼ばれる。この中でPCとの通信の送受信を行う。
- event_handler()
  - 何かイベントが発生したときに呼ばれる。内容は未実装
- receive()
  - timer_handlerの中で呼ばれる受信部分。ここでPCからのスタートやストップのイベントを受信したら、EventControllerへイベントを通知する
- send()
  - マーカ検出の実行。PCへの送信を行う
  - 注）マーカー検出はこの関数内で行なっています。今まで誤検出はありませんが、もし、誤検出する場合はこの関数を修正してください

マーカー検出について（イメージ）

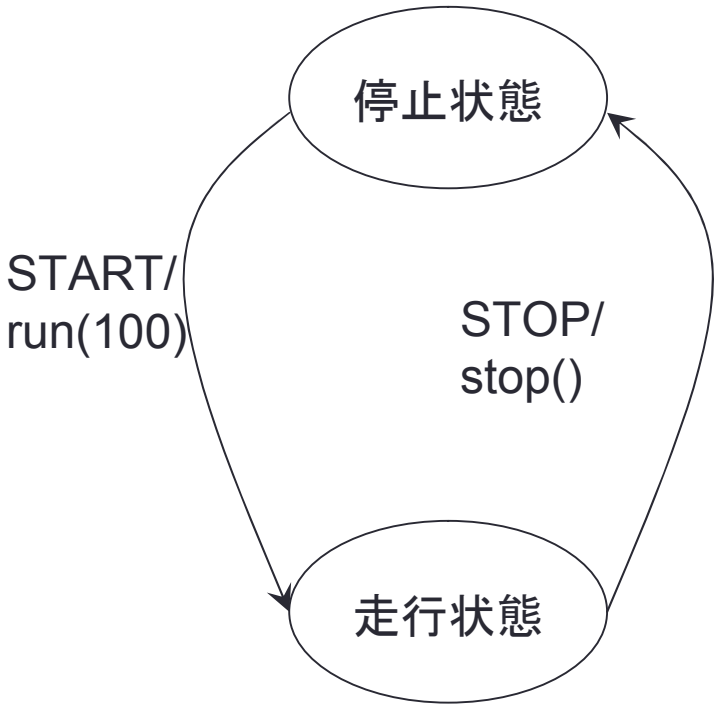


MotorController



- モータを制御するクラス。上位クラスは、Handler
- init()
  - 初期化
- pid()
  - PID計算を実行し、補正値を得る
- **turn()**
  - **PIDの補正値を車輪の回転へ変換する部分。この競技の肝となる部分。このプログラムを修正することで、よりなめらかな動きが実現できる。**
- run()
  - 右と左のモータをスタートさせる。スピードの値は引数によって渡される。
- stop()
  - 右と左のモータをストップさせる
- state_machine()
  - 状態遷移関数。イベントは、「スタート」、「ストップ」の2つ。状態は、「止まっている」「止まっていない」の2つこの関数は、event_handler()の中で呼ばれる。
- event_handler()
  - この関数はPcManagerのreceive()関数からイベントが発生した場合に呼ばれる。
- func_xxxxx()
  - state_machine()から呼ばれる関数（説明は割愛する）

MotorController内の状態遷移図



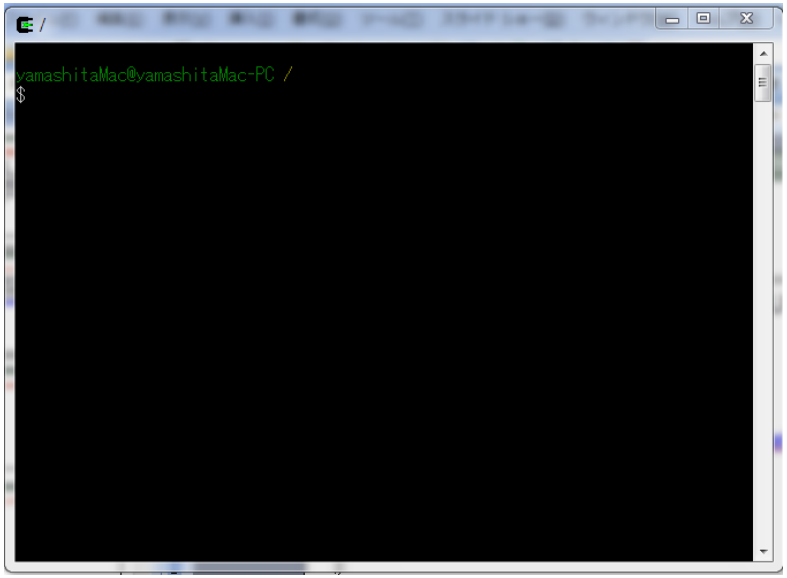
- このような簡単なシステムでも状態遷移を用いる理由として
  - 同じイベントが2度続いてくる場合の対処
    - STARTが2回きてもきちんと考えられてる
  - 状態や、イベントが仕様変更により増減しても対処が比較的楽にできる。
    - 状態遷移で設計した場合はコードへの変換がルーチンワーク的にできる

3. 開発環境

- cygwin
- nxtOSEK
- ディレクトリ構造
- Makefileの使い方
- ライブラリファイルの作り方
- プログラムアップロード
- 実行

cygwin

- Microsoft Windows OS上で動作するUNIXライクな環境であり、フリーソフト
- nxtOSEKを使うために必要
- インストール方法
  - http://lejos-osek.sourceforge.net/jp/installation_enf.htm を参照



nxtOSEK

- オープンソースのLEGO MINDSTOMES NXT用の開発/実行環境
- leJOS NXTという開発環境に含まているI/Oドライバ及びTOPPERSプロジェクトの成果物の一つである、TOPPERS/ATK、TOPPERS/JSPをNXTのハードへ移植したリアルタイムOSで構成されている
  - gccツールチェインを使用して、ANSI　C/C++言語が利用可能
  - NXTモータ、センサーのデバイスへの制御APIを用意
  - 自動車電子制御用のOSEKに対応したマルチタスクスケジューリング
  - UITRON4.0に対応したマルチタスクスケジューリング

ディレクトリ構造

- cygwin
  - nexttool
  - nxtOSEK
  - ユーザディレクトリ / (例： GETC_1)
    - ユーザ用プロジェクトディレクトリ / (例：getc2011_sample_cpp)
      - ユーザ用ソースコード　 xxx.cpp xxx.h

Makefileの書き方

```
# nxtOSEKルートディレクトリ
NXTOSEK_ROOT = ../../nxtOSEK

# ターゲット実行形式ファイル名
TARGET = getc2011sample

# インクルードパス
#USER_INC_PATH=
   $(NXTOSEK_ROOT)/ecrobot/nxtway_gs
   _balancer
USER_INC_PATH= ../

# ライブラリ
#USER_LIB = nxtway_gs_balancer
USER_LIB = keisoku

# Cソースファイル
TARGET_SOURCES = ¥
```

```
# C++(.cpp)ソースファイル
TARGET_CPP_SOURCES = ¥
   pc_manager.cpp ¥
   display.cpp ¥
  bluetooth.cpp ¥
  motor_controller.cpp ¥
  nxt.cpp ¥
   event_controller.cpp ¥
   timer_controller.cpp ¥
   getc2011_sample.cpp ¥
  manager.cpp
# TOPPERS/ATK(OSEK)設定ファイル
TOPPERS_OSEK_OIL_SOURCE =
   getc2011_sample.oil
# 下記のマクロは変更しないでください
O_PATH ?= build
# C++(.cpp)ビルド用makefile
include
   $(NXTOSEK_ROOT)/ecrobot/ecrobot++.
   mak
```

Makeの実行

$**make all**
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/alarm.c to alarm.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/event.c to event.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/interrupt.c to interrupt.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/osctl.c to osctl.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/resource.c to resource.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/task.c to task.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/kernel/task_manage.c to task_manage.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/cpu_config.c to cpu_config.o
Compiling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/lego_nxt/sys_config.c to sys_config.o
Generating OSEK kernel config files from getc2011_sample.oil

中略

Assembling ../../nxtOSEK/ecrobot/../lejos_nxj/src/nxtvm/platform/nxt/vectors.s to vectors.o
Assembling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/debug.S to debug.o
Assembling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/lego_nxt/sys_support.S to sys_support.o
Assembling ../../nxtOSEK/ecrobot/../ecrobot/c/nxt_binary_header.s to nxt_binary_header.o
Assembling ../../nxtOSEK/ecrobot/../ecrobot/c/nxt_entry_point.s to nxt_entry_point.o
Assembling ../../nxtOSEK/ecrobot/../ecrobot/c/ecrobot_init.s to ecrobot_init.o
Assembling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/cpu_support.S to cpu_support.oram
Assembling ../../nxtOSEK/ecrobot/../toppers_osek/config/at91sam7s-gnu/irq.s to irq.oram
Generating binary image file: getc2011sample_rom.bin
Generating binary image file: getc2011sample_ram.bin
Generating binary image file: getc2011sample.rxe

Makeのクリーン

- make clean
- 上記コマンドを実行すると、makeによって生成されたオブジェクトや必要のないファイル、実行ファイルがすべて削除される。

- make allを行なって、どうしてもコンパイルが通らない場合は、一度make cleanを行なって、再度make allを行うと通る場合がある。
  - これは、あるファイルを修正したら、影響のあるファイルをコンパイルしなければならないのにもかかわらず（依存関係）コンパイルしてなかったという現象がたまに起こるからである。
  - この現象はUNIXではMakefileの修正を行うことで対策できるのだが、今回紹介しているMakefileではその修正が困難

ライブラリファイルの作り方

ディレクトリ構造

- cygwin
  - nexttool
  - nxtOSEK
  - ユーザディレクトリ / (例： GETC_1)
    - ライブラリ用ディレクトリ / (例：lib)
      - アーカイブ名ディレクトリ / (例：keisoku)
        - ライブラリ関数 xxx.c xxx.h
    - ユーザ用プロジェクトディレクトリ / (例：getc2011_sample_cpp)
      - ユーザ用ソースコード　 xxx.cpp xxx.h

ライブラリファイルの作り方（Makefile)

```
# Makefile for NXTway-GS balancer library

# modified to support new directory structure by takshic
ROOT := $(dir $(lastword $(MAKEFILE_LIST)))../../../nxtOSEK

ECROBOT_ROOT = $(ROOT)/ecrobot
# added to support new directory structure by takshic
ECROBOT_C_ROOT = $(ECROBOT_ROOT)/c

LEJOSNXJSRC_ROOT = $(ROOT)/lejos_nxj/src/

LEJOS_PLATFORM_SOURCES_PATH =
    $(LEJOSNXJSRC_ROOT)/nxtvm/platform/nxt
LEJOS_VM_SOURCES_PATH = $(LEJOSNXJSRC_ROOT)/nxtvm/javavm

C_LIB_SOURCES = ¥
    send_bt.c

C_OPTIMISATION_FLAGS = -Os
include $(ECROBOT_ROOT)/tool_gcc.mak

INC_PATH := ¥
    $(LEJOS_PLATFORM_SOURCES_PATH) ¥
    $(LEJOS_VM_SOURCES_PATH) ¥
    $(ECROBOT_ROOT) ¥
    $(ECROBOT_C_ROOT)

O_FILES = $(C_LIB_SOURCES:c=o)

TARGET = ../../libkeisoku.a

.PHONY: all
all: $(TARGET)

$(TARGET): $(O_FILES)
    @echo "Creating $@"
    $(AR) rv $(TARGET) $(O_FILES)

%.o: %.c
    @echo "Compiling $< to $@"
    $(CC) $(CFLAGS) -o $@ $<

%.oram: %.c
    @echo "Compiling $< to $@"
    $(CC) $(CFLAGS) -o $@ $<

%.o: %.s
    @echo "Assembling $< to $@"
    $(AS) $(ASFLAGS) -o $@ $<

%.oram: %.s
    @echo "Assembling $< to $@"
    $(AS) $(ASFLAGS) -o $@ $<

%.obmp : %.bmp
    @echo "Converting $< to $@"
    $(OBJCOPY) -I binary -O elf32-littlearm -B arm ¥
    $< $@

.PHONY: release
release:
    rm $(O_FILES)

.PHONY: clean
clean:
    rm $(TARGET)
    rm $(O_FILES)
```

プログラムアップロード

- USBケーブルを使って、開発用PCとNXT本体を接続する
- 初めて接続する場合は、USBケーブルが接続されたというメッセージがでるので、それまで待つ（OSによって表示が変わる）
- NXTの電源をONにする
- Makeしたディレクトリに移動する
- 次のコマンドを打つ
  - $sh rxeflash.sh ($はcygwinのプロンプトなので、打たない）

プログラムアップロード

- 次のメッセージがでてきたらOK
  - Executing NeXTTool to upload getc2011sample.rxe...
  - getc2011sample.rxe=30336
  - NeXTTool is terminated.
    - ファイル名や、サイズはプログラムによって変化します
- この時に、getc2011sample.rxe=30336が表示されない場合はアップロードに失敗したと考えたほうが良い
  - 失敗する主な原因は次の通り
    - NXTの電源が入っていない場合
    - PCへUSBケーブルが接続されていない場合
    - NXTでプログラムが実行されている場合（これが一番多いので注意）

libkeisoku.aの説明

- ライブラリとは
  - libxx.aとあるのは、一般的に関数コンパイルしたものをアーカイブしたものである。
  - 単体では動作できないので実行ファイルではない。
  - arコマンドでアーカイブされるので、拡張子がaになっている
  - Make時にリンクされるので、静的ライブラリとも呼ばれる。
  - 組み込みシステムでは、動的ライブラリの置く場所がもったいないので、静的ライブラリの活用がほとんど

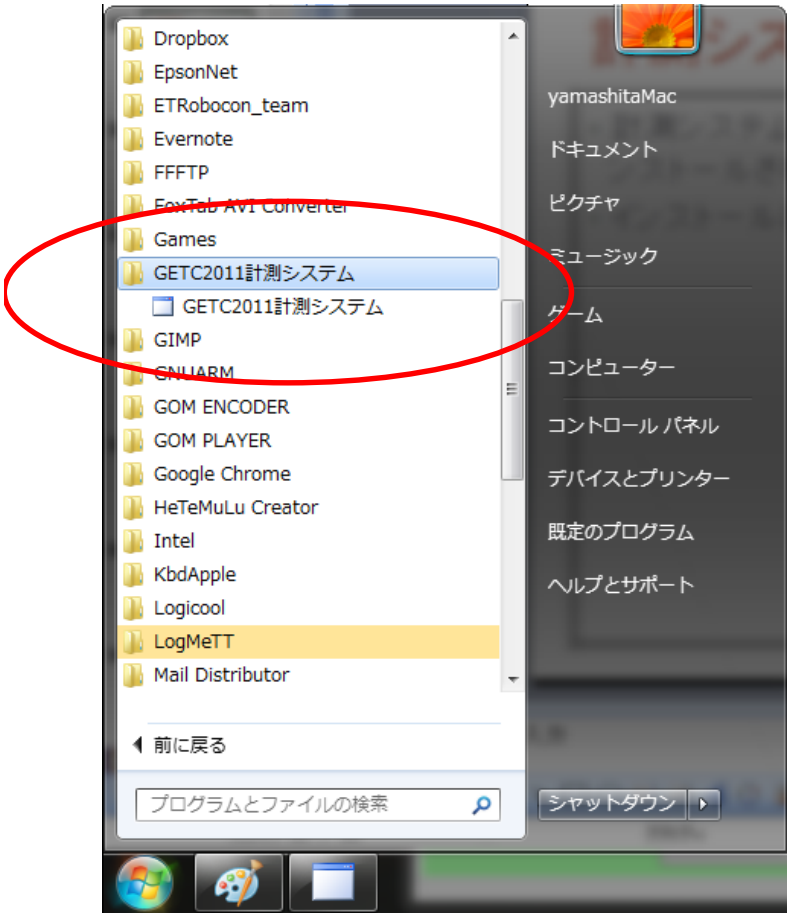
- libkeisoku.aに含まれる、void send_bt()関数の説明

```
//説明：interval_time/reduce_time毎に電圧の平均をとり，light_dataの値とともに
//      計測器へ送信する．送信が終わったら，light_dataは0にセットされる
//引数説明 int interval_time Bluetooth送信間隔 msec     値変更不可
//引数説明 int reduce_time   PcManagerへの通知間隔 msec 値変更不可
//引数説明 int *light_data  マーカ検出値 1:検出 0:未検出。ライブラリの中で0に設定される。
//引数説明 U8  data1,data2  ユーザデータ
//引数説明 S32 data21,data22,data23,data24 ユーザーデータ
//引数説明 S16 data31,data32,data33        ユーザーデータ
 void send_bt(int interval_time , int reduce_time , int *light_data ,
                              U8 data11 , U8 data12,
                              S32 data21, S32 data22,S32 data23, S32 data24,
                              S16 data31 , S16 data32 ,S16 data33  );
```

4. 実行手順

- 計測システムのインストール方法
- Bluetoothについて
- NXTのBluetoothの設定
- PCとNXTのBluetoothのコネクション
- COMポート番号の確認
- プログラム実行からコース設置までの手順
- 計測システムの使用方法
- 既知のバグ

計測システムのインストール



- 計測システムは、setup.exeファイルを実行すると自動的にインストールされる
- インストールされた計測システムは、スタートメニューに登録される
- アンインストールは、コントロールパネルから通常のアプリケーションと同様に行う。
- 新しいリリースをインストールする場合は、古いバージョンは必ずアンインストールすること

Bluetoothについて

- Bluetoothは、PCに内蔵されているものや、市販のものが利用できる
- よく知られている問題として、Bluetoothに付属しているTOSHIBAのドライバをインストールすると、NXTとの通信ができないらしい
  - ドングル等に付属しているドライバーはあえてインストールしなくても、Windowsに付属しているマイクロソフトのドライバーが自動的にインストールされる
  - 試していないがLEGO社が販売している公式のBluetoothドングルも利用できると考えられる。（ETロボコンでは推奨ドングル）
  - 下記の事例で動作確認しています
    - Mac内蔵のBluetooth＋bootcamp＋Windows7 Home Premium
    - Corega製マイクロサイズブルートゥースUSBアダプタ＋Panasonic CW-7 Windows Vista

NXTのBluetooth設定

- 電源が入ったあとのTOPメニューから左右どちらかの矢印ボタンを3回押して、Bluetoothの設定メニューを出し、ENTERキーを押下する
- BluetoothをONするために、そのままENTERキーを押下する

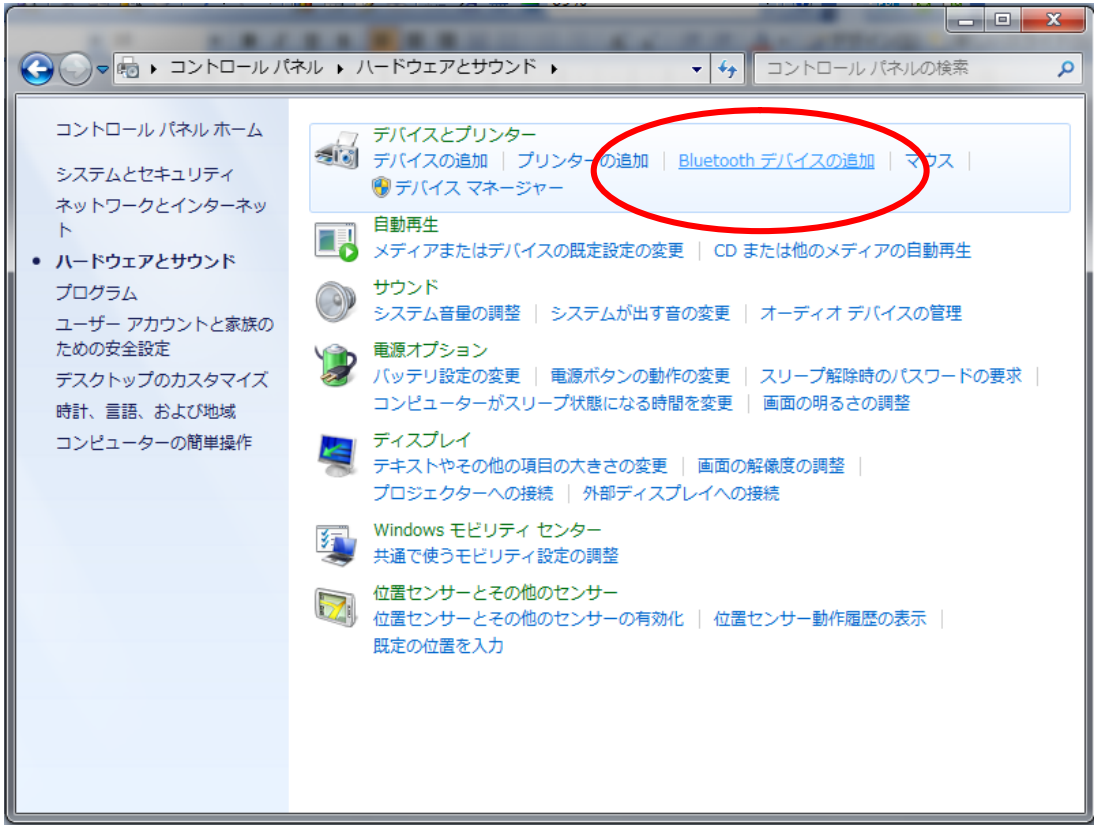
NXTのBluetooth設定

- ONすると、左上にBluetoothのマークが点灯する
- 左右ボタンを複数回押下してVisibilityを表示させる
- ENTERキーを2回押下し、左上にブルートゥースマークの横に、“<”が表示されれば、OK
- この設定は1回行うだけでよい。
- このマークが消えた場合は再度行う必要がある

PCとNXTのBluetoothのコネクション

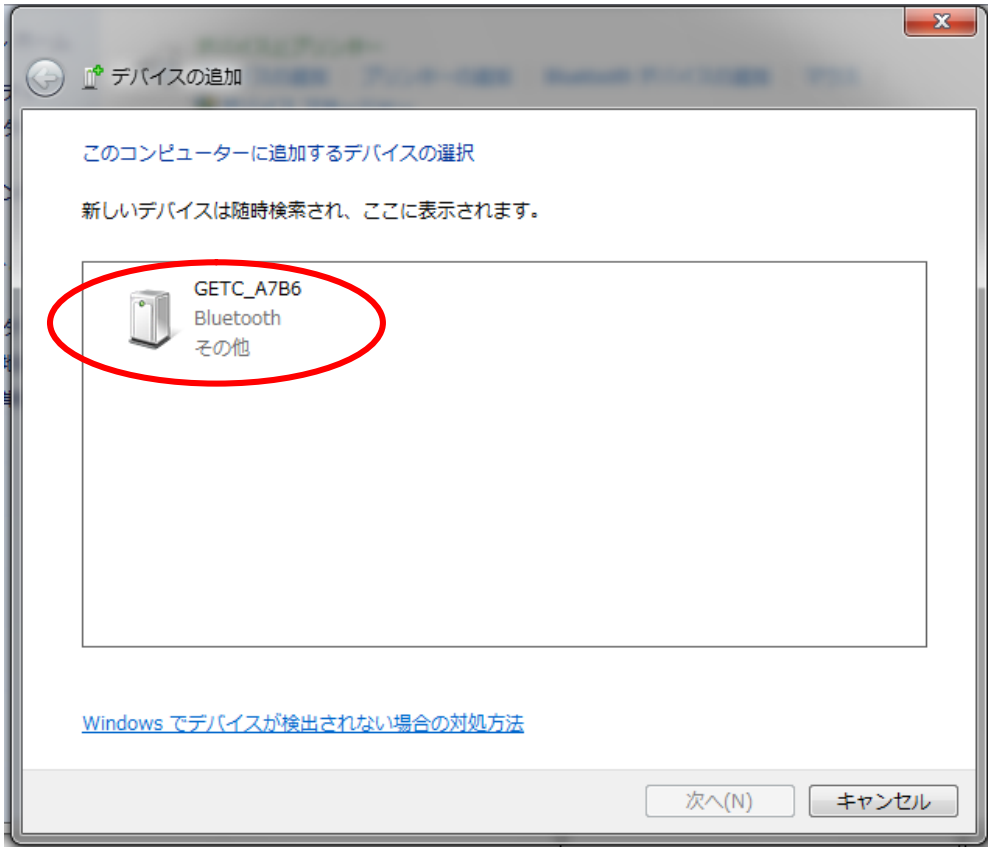
- Windows7の場合
  - スタートーコントロールパネルを開く
  - ハードウェアとサウンドの中にある「デバイスの追加」をクリック

PCとNXTのBluetoothのコネクション



- Windows7の場合
  - 「Bluetoothデバイスの追加」をクリック
  - NXTの電源を入れる

PCとNXTのBluetoothのコネクション

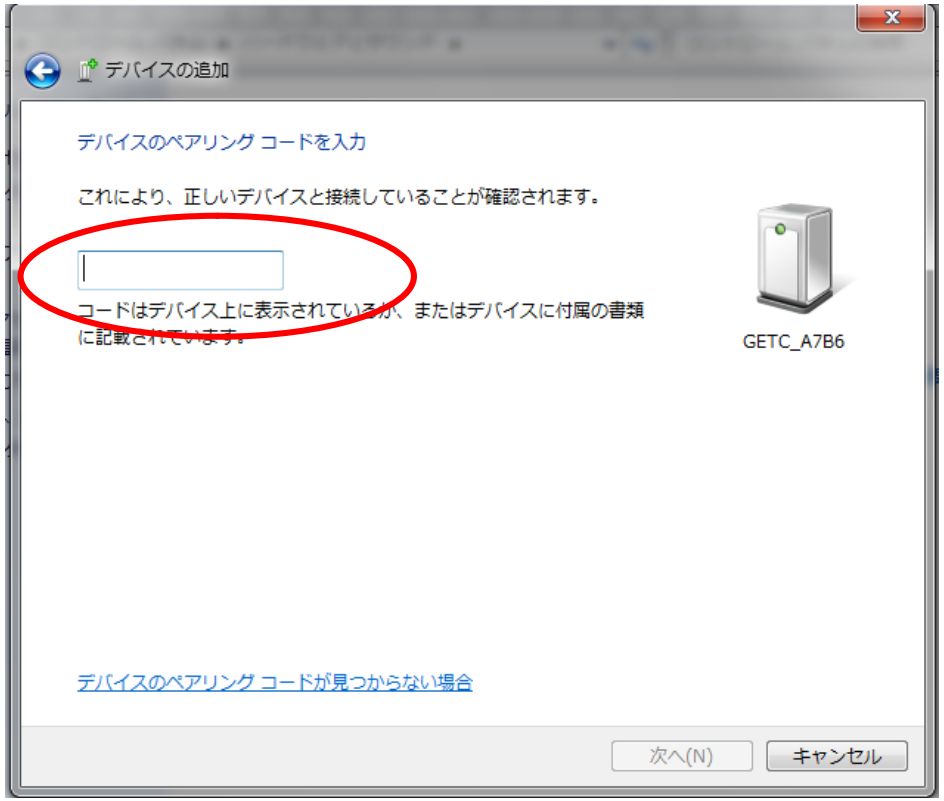


- Windows7の場合
  - デバイスの追加画面がでる
  - しばらくすると、NXTが検出される
  - 検出されたNXTをダブルクリックする

PCとNXTのBluetoothのコネクション

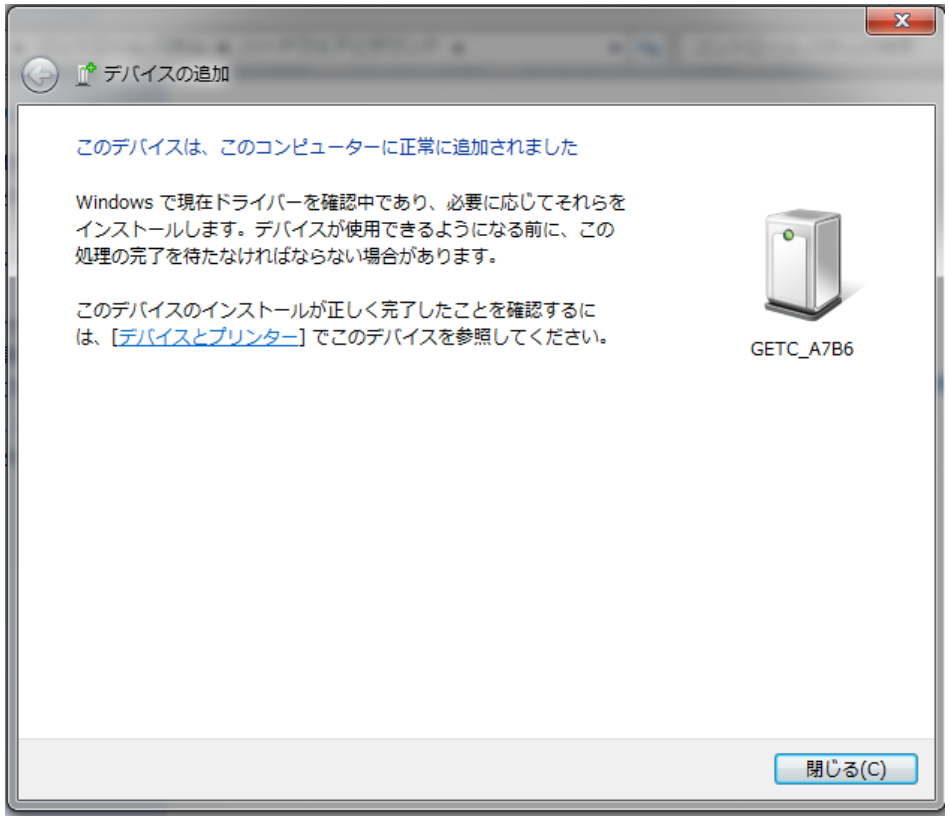
- Windows7の場合
  - NXT側の画面が切り替わりパスキーの入力を促される
  - ENTERキーの下の灰色キーでクリア
  - 矢印キーでパスキーの選択
  - ENTERキーで決定
  - ✓を選択すると入力終了となる
  - 1234というパスキーはデフォルトになっているので、適当な4桁の数字を入れる
  - 大会時のパスキーは運営から提示される

PCとNXTのBluetoothのコネクション



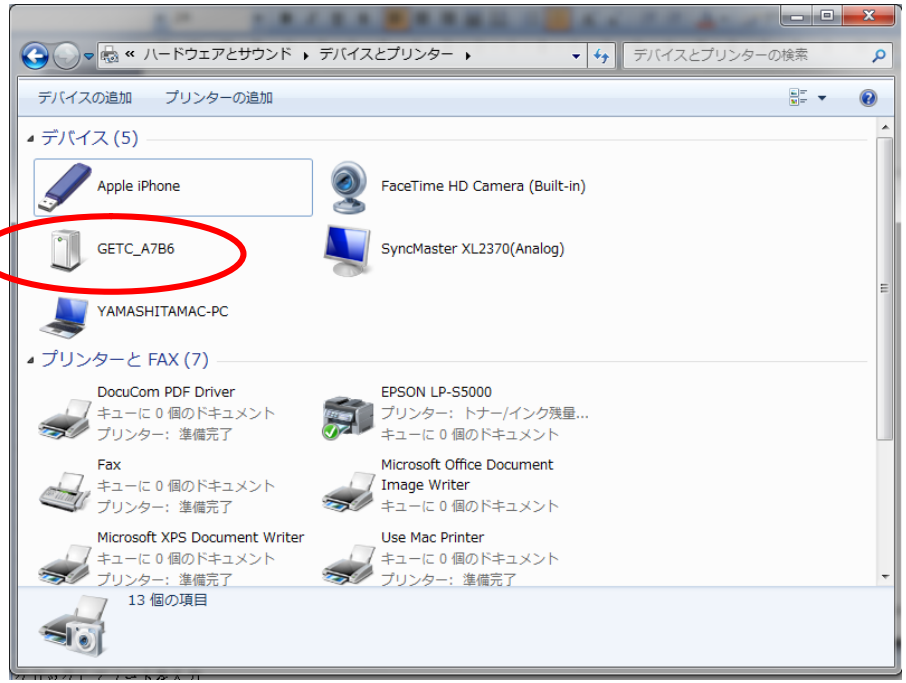
- Windows7の場合
  - NXTでパスキーを入れるとWindowsに左図のような画面が出る。
  - ここで、先程NXTへ入力した4桁の数字を入れる

PCとNXTのBluetoothのコネクション



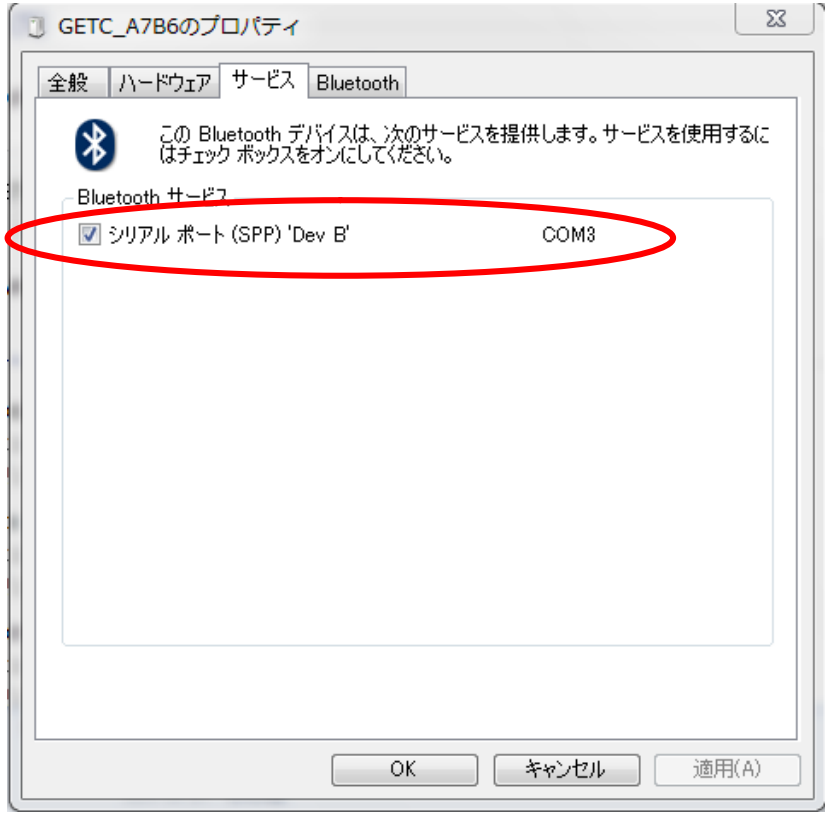
- Windows7の場合
  - 左図の画面がでれば登録OK「閉じる」を押下

COMポート番号の確認



- Windows7の場合
  - スタートーコントロールパネルーハードウェアとサウンドーデバイスとプリンターをクリック
  - 画面に追加したNXTがあるので、ダブルクリック

COMポート番号の確認



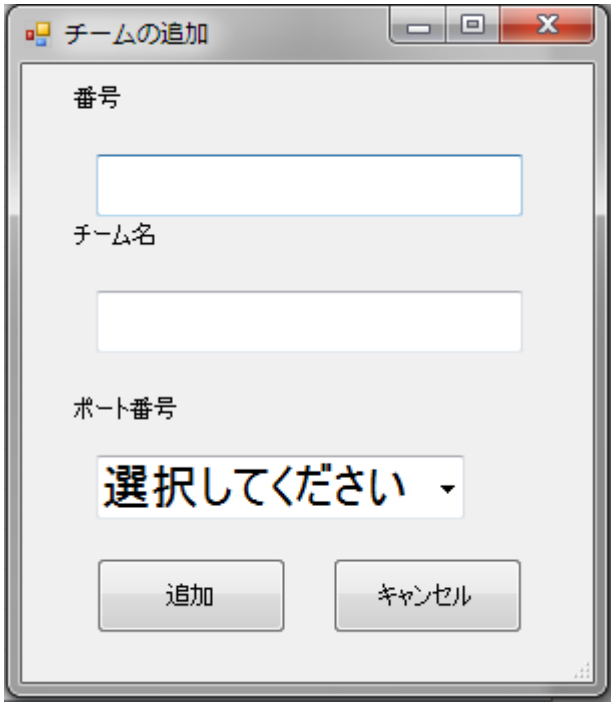
- Windows7の場合
  - プロパティ画面がでるので、「サービス」タブをクリックすると、COMポートの番号が確認できる。
  - 注意）Windows7はNXTを削除、追加を繰り返してもこの番号は変わらない。
  - Windows Vistaは、削除、追加を繰り返すとポート番号が増える一方になり、その番号を戻すには手こずるので注意

計測システムの使用方法

計測システム:チームの登録

- 設定タブをクリックすると設定画面が表示される
- 追加ボタンを押下すると、チームが登録できる

計測システム:チームの登録



- 追加画面で、
  - 番号:チームID
  - チーム名:チーム名
  - ポート番号:上記で調べたCOMポート番号
- を入力して追加ボタンを押下する。

計測システム:チームの登録

- チームを東へ登録する場合は、リスト上のチームを選択して、「東へ追加」を押下する

計測システム:チームの登録

- 左上の計測システムのタグをクリックすると
- 東へチーム名が登録される。

プログラム実行からコース設置までの手順

- 電源を入れる
- 電源投入は、橙色のENTERボタンで行う
- 中央のにMyFilesがあればENTERキーを押下
  - なければ、灰色の矢印キーで中央にもってくる

プログラム実行からコース設置までの手順

- 中央にSoftware filesがあれば、ENTERキーを押下
- 自分で作成したプログラム名が中央にあれば、ENTERキーを押下
- Runが中央にあれば、ENTERキーを押下

プログラム実行からコース設置までの手順

- プログラムが実行して一時停止するので、RUNをするために右矢印キーを押下
- ここから、実際のプログラムが走るので、自分が作成した通りに操作する。Demoプログラムの動きを下記に示す
  - 「黒の上に置け」とメッセージが出るので、マーカー以外のコース場所において、ENTERキーを押下
  - 「白の上に置け」とメッセージが出るので、マーカーの上において、ENTERキーを押下
  - 最後に「Bluetoothを待っている」とメッセージがでるので、この状態でコースに設置する

プログラム実行からコース設置までの手順

- NXTの準備が整ったら、計測システムとNXTとのコネクションを行う。
  - 左上のOPENボタンを押下
  - しばらくすると、「ポート閉」ー「ポート開待機」ー「停止中」と状態が変化する
  - この時間は数秒を要する
  - 「停止中」であれば、左下の「START」ボタンで走行体が走りだす
  - 状態は「走行中」に変化する
  - 止める場合は、東あるいは西の強制ストップボタンを押下する
  - 画面コピーボタンを押下すると画面のハードコピーを得ることができる

既知のバグ

- 計測システムにおいて、「画面クリア」を行なってもクリアしない場合がある。
  - 対策：アプリの再起動
- 「ポート開待機」状態で「停止中」へ遷移しない場合がある
  - 対策：アプリの再起動及びNXTの再キャリブレーションが必要

大会まで時間が足りません

- 大丈夫です
  - まずは、基本プログラムで動作確認してください
  - turn()関数を改良してください。
    - この関数を改良することにより、Demoプログラムは大幅に改良されます。
  - PID制御で利用している定数kp,ki,kdを変更してみてください
    - PID制御用のこれらの定数は大変デリケートです。
    - この定数の変化に伴い、補正値が大きく変わってきます。
    - この補正値をグラフ化することにより、PIDの見える化ができますので、デバッグ機能を使ってチャレンジすると新しい発見があるかもしれません。

余裕でクリアしました

- 時間が余ったら
  - Demoプログラムを設計変更してみてください。
    - 悪いところを改良
    - 気になったところを改良
    - 性能的なところを改良
    - 状態遷移を改良（徐々にスタート等）
    - マルチタスク化等に挑戦

おわりに

- PBLを実践することにより、計画的に、生産的に皆様の設計ノウハウが向上することを願います。

ご清聴ありがとうございました。

修正履歴

日付	修正者	内容	備考
2011/10/6	山下	・初版	1.0.0