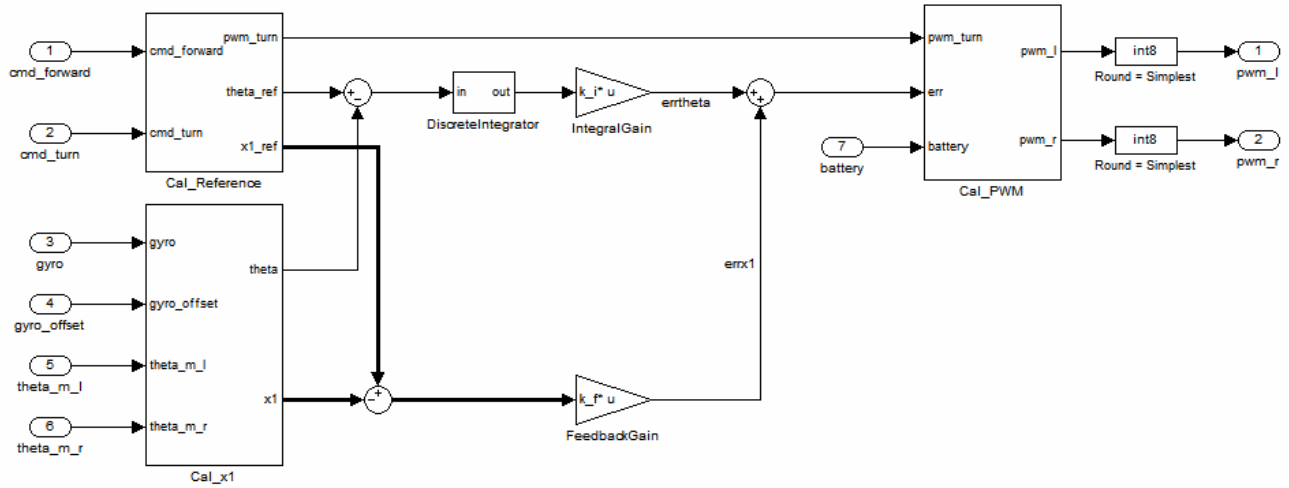
# NXTway-GS 倒立振子制御 C API 解説書




# はじめに

## 本文書について

本文書は、LEGO® MINDSTORMS® NXT を使用した 2 輪倒立振子ロボット(NXTway-GS)に搭載されている倒立振子(とうりつしんし)制御 C API の解説資料です。本文書は、組込みソフトウェア技術者を対象に、倒立振子制御 C API の開発手法や使用方法を紹介するものであり、倒立振子制御に用いられている制御理論等については触れていません。倒立振子制御 C API に適用されている制御理論等の詳細については、下記の URL を参照してください。

NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design (山本順久)

http://www.mathworks.com/matlabcentral/fileexchange/19147

※上記 URL からダウンロードした ZIP ファイルには、Simulink モデル一式に加えて、倒立振子制御に適用した制御理論などの詳細な説明が記された日本語 PDF 資料「NXTway-GS のモデルベース開発」が含まれています。

## 免責事項

本文書に掲載されている情報により生じた損害について、著者はいかなる責任も負いかねますことを予めご了承ください。

## 商品名称について

LEGO®および MINDSTORMS®はレゴ社の登録商標です。また MATLAB® および Simulink® は The MathWorks,Inc.の登録商標です。その他、本資料に記載されている製品名とブランド名は、それぞれ該当する各社の商標または登録商標です。

# 目次

# 1 モデルベース開発

本章では、倒立振子制御 C API の開発に用いられている、モデルベース開発という開発手法の全般について概説します。

## 1.1 モデルベース開発とは

モデルベース開発（Model Based Design / Development、MBD と略されます）とは、シミュレーション可能なモデルを用いるソフトウェア開発手法です。制御系 MBD では、制御器および制御対象、またはその一部をモデルで表現し、机上シミュレーション／リアルタイムシミュレーションにより制御アルゴリズムの開発・検証を行います。リアルタイムシミュレーションとは、制御系の一部を実機、その他をリアルタイムシミュレータ上で動作するモデル生成コードとし、実時間での動作検証を行うシミュレーション技術のことです。さらに、Real-Time Workshop® Embedded Coder 等の C コード生成ツールを用いて、制御器モデルから実際の制御器（マイコン等）に組み込む制御用 C プログラムを作成することができます。図 1-1 は The MathWorks 社の MATLAB という製品を用いた場合の制御系 MBD の概念図です。

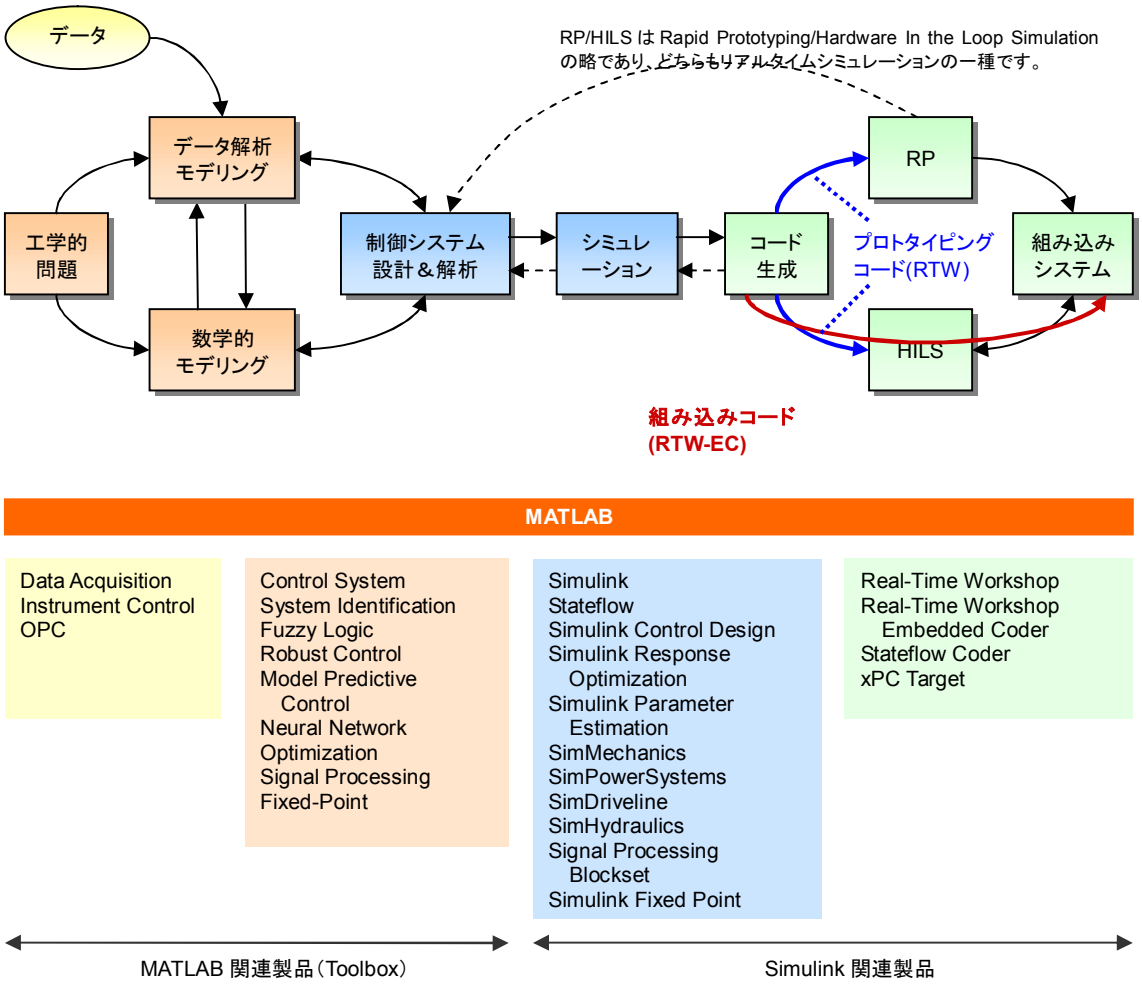



図 1-1 MATLAB 製品を用いた制御系 MBD

## 1.2 モデルベース開発プロセス

MBD による制御ソフトウェアの開発プロセスは、図 1-2 に示す V プロセスを用いて説明されます。V プロセスはソフトウェア開発を要求分析、各種設計、コーディングの工程に分け、各工程に検査（テスト）を対応させた V 字型の開発プロセスです。MBD では、V プロセスの左側の工程でモデリングを行い、制御仕様完成度の早期向上を図ります。また、作成したモデルを検証工程で利用することにより、コード品質・検証効率向上を図ります。

MBD には、以下のようなメリットがあります。

- 机上シミュレーションによる仕様ミスの早期検証
- リアルタイムシミュレーションによる試作工数削減・フェイルセーフ検証
- モデル検証機能によるテストの効率化
- モデル仕様の共通認識によるコミュニケーション改善
- 自動コード生成による人的コーディング工数・エラー削減

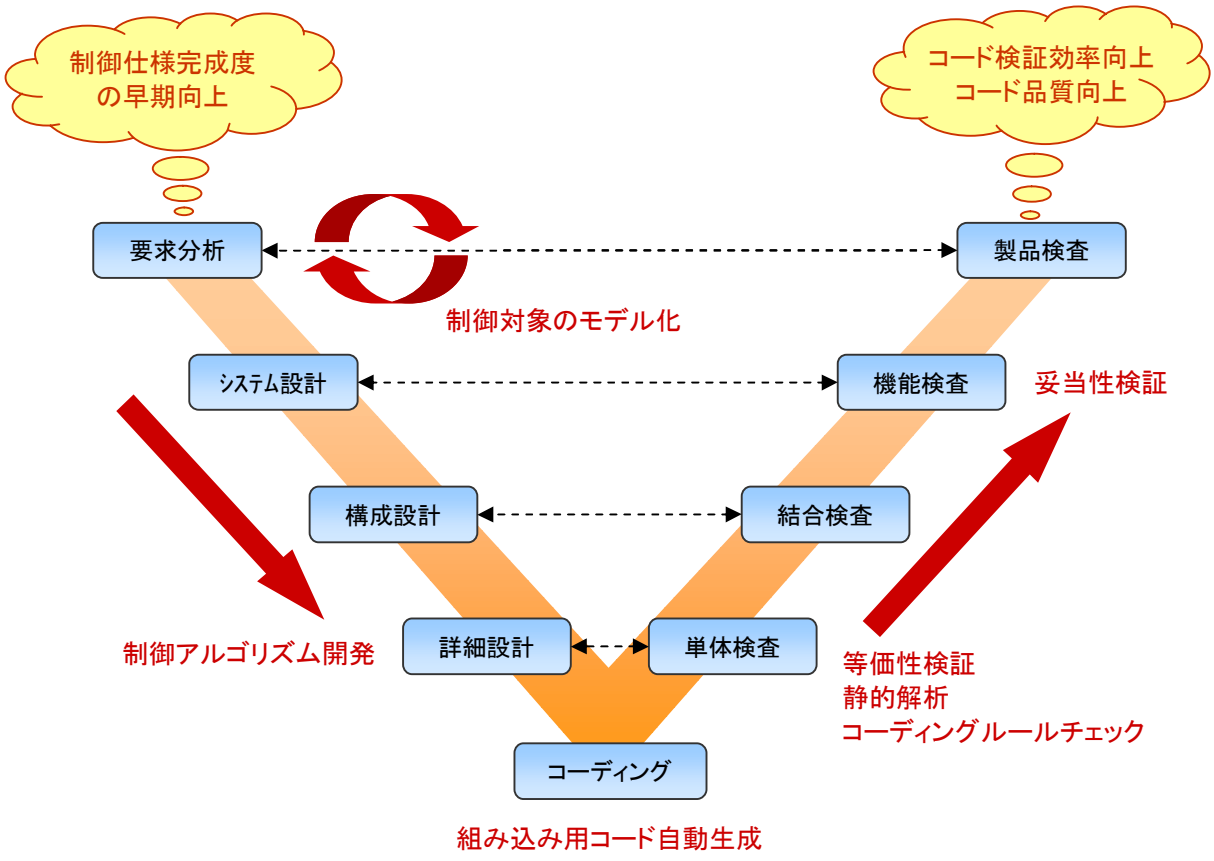



図 1-2 制御系 MBD の V プロセス

## 1.3 倒立振子制御 C API のモデルベース開発プロセス

倒立振子制御 C API の開発は、具体的には図 1-3 に示されている開発プロセスに沿っておこないました。開発プロセスを大別すると次の 4 つに分けることができます。

- 制御対象のモデル化: 制御対象である 2 輪倒立振子ロボットの物理的振る舞いを解析するための運動方程式の導出および 2 輪倒立振子ロボット物理モデルの作成
- 制御アルゴリズム設計(ラピッドプロトタイピング): 2 輪倒立振子ロボットを制御するための制御器の設計/評価。Simulink 上に構築された Embedded Coder Robot NXT という LEGO MINDSTORMS NXT 用モデルベース開発環境を用いた、2 輪倒立振子ロボット物理モデルと制御器のシミュレーションおよび制御器の実機検証(ラピッドプロトタイピング)
- 実装モデル設計: 制御アルゴリズム設計で開発された倒立振子制御アルゴリズムをベースに実装用モデルを設計
- 組み込み用コード生成: 実装モデルから Real-Time Workshop Embedded Coder という C コード生成ツールを用いて、倒立振子制御 C API を自動生成

Simulink 上でのモデルベース開発の様子については、下記の URL に動画が紹介されています。

http://www.youtube.com/watch?v=EHPlGTLQHRc

2 輪倒立振子ロボットのラピッドプロトタイピングの様子については、下記の URL に動画が紹介されています。

http://www.youtube.com/watch?v=4ulBRQKCwd4

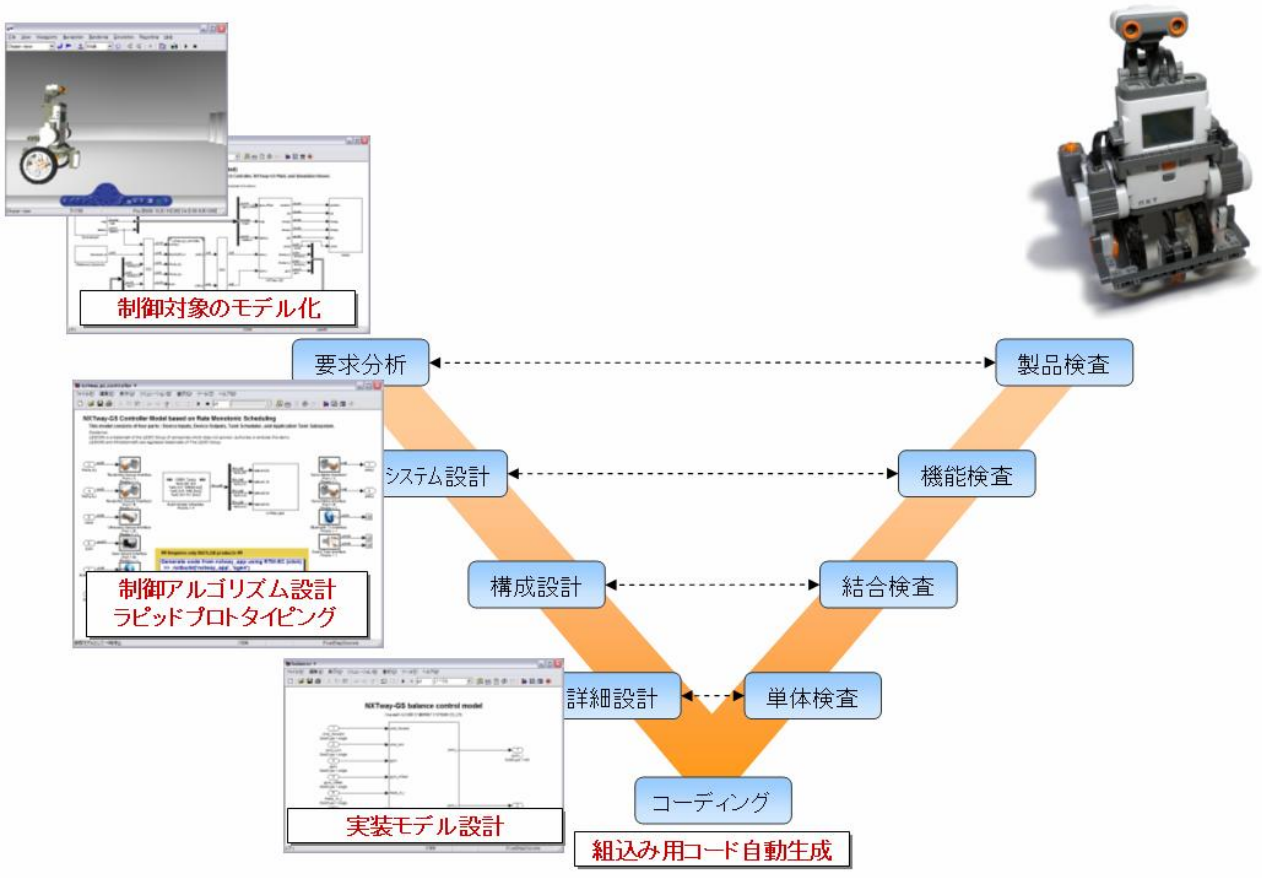



図 1-3 倒立振子制御 C API の開発プロセス

# 2 倒立振子制御 C API

本章では、倒立振子制御 C API の仕様について説明します。

## 2.1 API 仕様

倒立振子制御 C API は下記の 2 つの関数から構成されています。本 API は C ライブラリの形(libnxtway_gs_balancer.a)で nxtOSEK/ecrobot フォルダ内に格納されています。また C ソースファイルは nxtOSEK/ecrobot/nxtway_gs_balancer フォルダに格納されています。倒立振子制御 C API を使用する場合は、次の設定が必要になります。

- balancer.h ヘッダファイルをインクルードする
- Makefile の USER_INC_PATH マクロに nxtOSEK/ecrobot/nxtway_gs_balancer パスを定義する
- Makefile の USER_LIB マクロに nxtway_gs_balancer ライブラリを定義する

| 倒立振子制御 C API | 説明 |
|---|---|
| void<br>**balance_control**(<br>F32 *args_cmd_forward*,<br>F32 *args_cmd_turn*,<br>F32 *args_gyro*,<br>F32 *args_gyro_offset*,<br>F32 *args_theta_m_l*,<br>F32 *args_theta_m_r*,<br>F32 *args_battery*,<br>S8 **ret_pwm_l*,<br>S8 **ret_pwm_r*) | 倒立振子制御の実行関数。この関数は 4msec 周期で起動されることを前提に設計されています。<br>※ジャイロセンサオフセット値は角速度 0[deg/sec]時のセンサ値であり、センサ個体差および通電によるドリフトを伴いますので、適宜補正する必要があります。また、左右の車輪駆動モータは個体差により、同じ PWM 出力を与えても回転数が異なる場合があります。その場合は別途補正機能を追加する必要があります。<br><br>**引数**:<br>*args_cmd_forward*: 前後進命令値。 100(前進最大)～-100(後進最大)<br>*args_cmd_turn*: 旋回命令値。 100(右旋回最大)～-100(左旋回最大)<br>*args_gyro*: ジャイロセンサ値<br>*args_gyro_offset*: ジャイロセンサオフセット値<br>*args_theta_m_l*: 左モータエンコーダ値[度]<br>*args_theta_m_r*: 右モータエンコーダ値[度]<br>*args_battery*: バッテリ電圧値[mV]<br>**戻り値**:<br>*ret_pwm_l*: 左モータ PWM 出力値<br>*ret_pwm_r*: 右モータ PWM 出力値 |
| void<br>**balance_init**(void) | 倒立振子制御の初期化関数。内部状態量変数を初期化します。 この関数により倒立振子制御機能を初期化する場合は、併せて左右の車輪駆動モータのエンコーダ値を 0 にリセットしてください。 |

※データタイプ F32: float, S8: signed char

nxtOSEK/ecrobot/nxtway_gs_balancer フォルダに格納されている balancer.c ファイルに倒立振子制御アルゴリズムが実装されています。balancer.c ファイルは自動生成 C ソースファイルであり、設定により任意のデータ定義やコメントが実装されています。(下記のサンプルでは、紙面の都合上、一部コメントを削除しています)

```
/**
 ******************************************************************************
 **   ファイル名 : balancer.c
 **
 **   概要       : 2輪型倒立振子ロボット「NXTway-GS」バランス制御プログラム
 **                NXTway-GSのバランス制御には、サーボ制御(状態 + 積分フィードバック)
 **                という現代制御を適用しています。制御対象の同定および制御器の開発
 **                にはThe MathWorks社のMATLAB&Simulinkという製品を使用した、
 **                MBD(モデルベースデザイン/開発)開発手法を用いています。このCプログラムは
 **                SimulinkモデルからReal-Time Workshop Embedded Coderコード生成標準機能を
 **                使用して自動生成されたものです。バランス制御器の制御パラメータについては
 **                ユーザーハンドコード側で定義する必要があります。定義例として、
 **                nxtOSEK¥samples¥nxtway_gs¥balancer_param.cを参照してください。
 **
 ** モデル関連情報:
 **   モデル名   : balancer.mdl
 **   バージョン : 1.893
 **   履歴       :  -
 **                 -
 **
 ** Copyright (c) 2008 CYBERNET SYSTEMS CO.,LTD.
 ** All rights reserved.
 ******************************************************************************
 **/
#include "balancer.h"
#include "balancer_private.h"

/*==============================================================================
 * Local macro definitions
 *============================================================================*/

/*==============================================================================
 * Data definitions
 *============================================================================*/
static F32 ud_err_theta;         /* 左右車輪の平均回転角度(θ)目標誤差状態値 */
static F32 ud_psi;               /* 車体ピッチ角度(φ)状態値 */
static F32 ud_theta_lpf;         /* 左右車輪の平均回転角度(θ)状態値 */
static F32 ud_theta_ref;         /* 左右車輪の目標平均回転角度(θ)状態値 */
static F32 ud_thetadot_cmd_lpf;  /* 左右車輪の目標平均回転角速度(dθ/dt)状態値 */
```

```
/*===========================================================================
 * Functions
 *===========================================================================*/

/* Model step function */
void balance_control(F32 args_cmd_forward, F32 args_cmd_turn, F32 args_gyro,
       F32 args_gyro_offset, F32 args_theta_m_l, F32 args_theta_m_r,
       F32 args_battery, S8 *ret_pwm_l, S8 *ret_pwm_r)
{
  {
    F32 tmp_theta;
    F32 tmp_theta_lpf;
    F32 tmp_pwm_r_limiter;
    F32 tmp_psidot;
    F32 tmp_pwm_turn;
    F32 tmp_pwm_l_limiter;
    F32 tmp_thetadot_cmd_lpf;
    F32 tmp[4];
    F32 tmp_theta_0[4];
    S32 tmp_0;

    tmp_thetadot_cmd_lpf = (((args_cmd_forward / CMD_MAX) * K_THETADOT) * (1.0F
      - A_R)) + (A_R * ud_thetadot_cmd_lpf);
    tmp_theta = (((DEG2RAD * args_theta_m_l) + ud_psi) + ((DEG2RAD *
      args_theta_m_r) + ud_psi)) * 0.5F;
    tmp_theta_lpf = ((1.0F - A_D) * tmp_theta) + (A_D * ud_theta_lpf);
    tmp_psidot = (args_gyro - args_gyro_offset) * DEG2RAD;

    tmp[0] = ud_theta_ref;
    tmp[1] = 0.0F;
    tmp[2] = tmp_thetadot_cmd_lpf;
    tmp[3] = 0.0F;
    tmp_theta_0[0] = tmp_theta;
    tmp_theta_0[1] = ud_psi;
    tmp_theta_0[2] = (tmp_theta_lpf - ud_theta_lpf) / EXEC_PERIOD;
    tmp_theta_0[3] = tmp_psidot;
    tmp_pwm_r_limiter = 0.0F;
    for (tmp_0 = 0; tmp_0 < 4; tmp_0++) {
      tmp_pwm_r_limiter += (tmp[tmp_0] - tmp_theta_0[tmp_0]) * K_F[(tmp_0)];
    }
    tmp_pwm_r_limiter = (((K_I * ud_err_theta) + tmp_pwm_r_limiter) /
      ((BATTERY_GAIN * args_battery) - BATTERY_OFFSET)) *  100.0F;

    tmp_pwm_turn = (args_cmd_turn / CMD_MAX) * K_PHIDOT;

    tmp_pwm_l_limiter = tmp_pwm_r_limiter + tmp_pwm_turn;
    tmp_pwm_l_limiter = rt_SATURATE(tmp_pwm_l_limiter, -100.0F, 100.0F);
    (*ret_pwm_l) = (S8)tmp_pwm_l_limiter;

    tmp_pwm_r_limiter -= tmp_pwm_turn;
    tmp_pwm_r_limiter = rt_SATURATE(tmp_pwm_r_limiter, -100.0F, 100.0F);
    (*ret_pwm_r) = (S8)tmp_pwm_r_limiter;

    tmp_pwm_l_limiter = (EXEC_PERIOD * tmp_thetadot_cmd_lpf) + ud_theta_ref;
    tmp_pwm_turn = (EXEC_PERIOD * tmp_psidot) + ud_psi;
    tmp_pwm_r_limiter = ((ud_theta_ref - tmp_theta) * EXEC_PERIOD) +  ud_err_theta;

    /* 次回演算用状態量保存処理 */
    ud_err_theta = tmp_pwm_r_limiter;
    ud_theta_ref = tmp_pwm_l_limiter;
    ud_thetadot_cmd_lpf = tmp_thetadot_cmd_lpf;
    ud_psi = tmp_pwm_turn;
    ud_theta_lpf = tmp_theta_lpf;
  }
}
```

```
/* Model initialize function */
void balance_init(void)
{
  ud_err_theta = 0.0F;
  ud_theta_ref = 0.0F;
  ud_thetadot_cmd_lpf = 0.0F;
  ud_psi = 0.0F;
  ud_theta_lpf = 0.0F;
}
```

## 2.2 制御パラメータ

倒立振子制御 C API の内部では、NXTway-GS の物理的振る舞いに基づいた制御用演算がおこなわれています。そのため、ロボットの組み立て方や使用するセンサ、モータの個体差の影響を受ける可能性があります。これらの個体差を補償するために、制御に必要なパラメータは別途、ユーザーが定義する必要があります。

| 制御パラメータ | 説明 |
|---|---|
| A_D | 左右車輪の平均回転角度用ローパスフィルタ係数。<br>データタイプ: F32<br>デフォルト値: 0.8 |
| A_R | 左右車輪の目標平均回転角度用ローパスフィルタ係数。<br>データタイプ: F32<br>デフォルト値: 0.996 |
| K_F[4] | 状態フィードバック係数。PID 制御の PD 制御係数に相当します。<br>K_F[0]: 車輪平均回転角度係数<br>K_F[1]: 車体傾斜角度係数<br>K_F[2]: 車輪平均回転角速度係数<br>K_F[3]: 車体傾斜角速度係数<br>データタイプ: F32<br>デフォルト値: {-0.870303, -31.9978, -1.1566, -2.78873} |
| K_I | サーボ制御用積分フィードバック係数。PID 制御の I 制御係数に相当します。<br>データタイプ: F32<br>デフォルト値: -0.44721 |
| K_PHIDOT | 車体目標旋回角速度係数。NXTway-GS 旋回方向の命令係数。<br>データタイプ: F32<br>デフォルト値: 25.0 |
| K_THETADOT | 車輪目標平均回転角速度係数。NXTway-GS 前後進方向の命令係数。<br>データタイプ: F32<br>デフォルト値: 7.5 |
| BATTERY_GAIN | PWM 出力算出用バッテリ電圧補正定数。<br>データタイプ: F32<br>デフォルト値: 0.001089 |
| BATTERY_OFFSET | PWM 出力算出用バッテリ電圧補正オフセット定数。<br>データタイプ: F32<br>デフォルト値: 0.625 |

※データタイプ F32: float

## 2.3 倒立振子制御 Simulink モデル

倒立振子制御 C API はハンドコードではなく、Simulink モデルから Real-Time Workshop Embedded Coder という C コード生成ツールにより自動生成されたものです。Simulink のライセンスが無くても Simulink モデルを閲覧できるように HTML ファイルに変換したものが、nxtOSEK/ecrobot/nxtway_gs_balancer/balancer_slwebview.html ファイルです。balancer_slwebview.html ファイルを SVG 対応 Web ブラウザから起動することで、図 2-1 の倒立振子制御 C API 用 Simulink HTML ファイルが開きます。

Simulink はブロック線図記述によるモデリング/シミュレーションソフトウェアです。モデルにおける矢印線(信号線)がデータの流れを示し、各ブロックは演算や処理の固まりを意味します。Simulink は制御、信号処理、画像処理などのアプリケーション領域において幅広く使用されており、DSL(Domain Specific Language)の一つともいえます。図 2-1 は Simulink モデルの最上位階層で、倒立振子制御における実行関数(balance_control)の引数/戻り値に対応した信号線が確認できます。

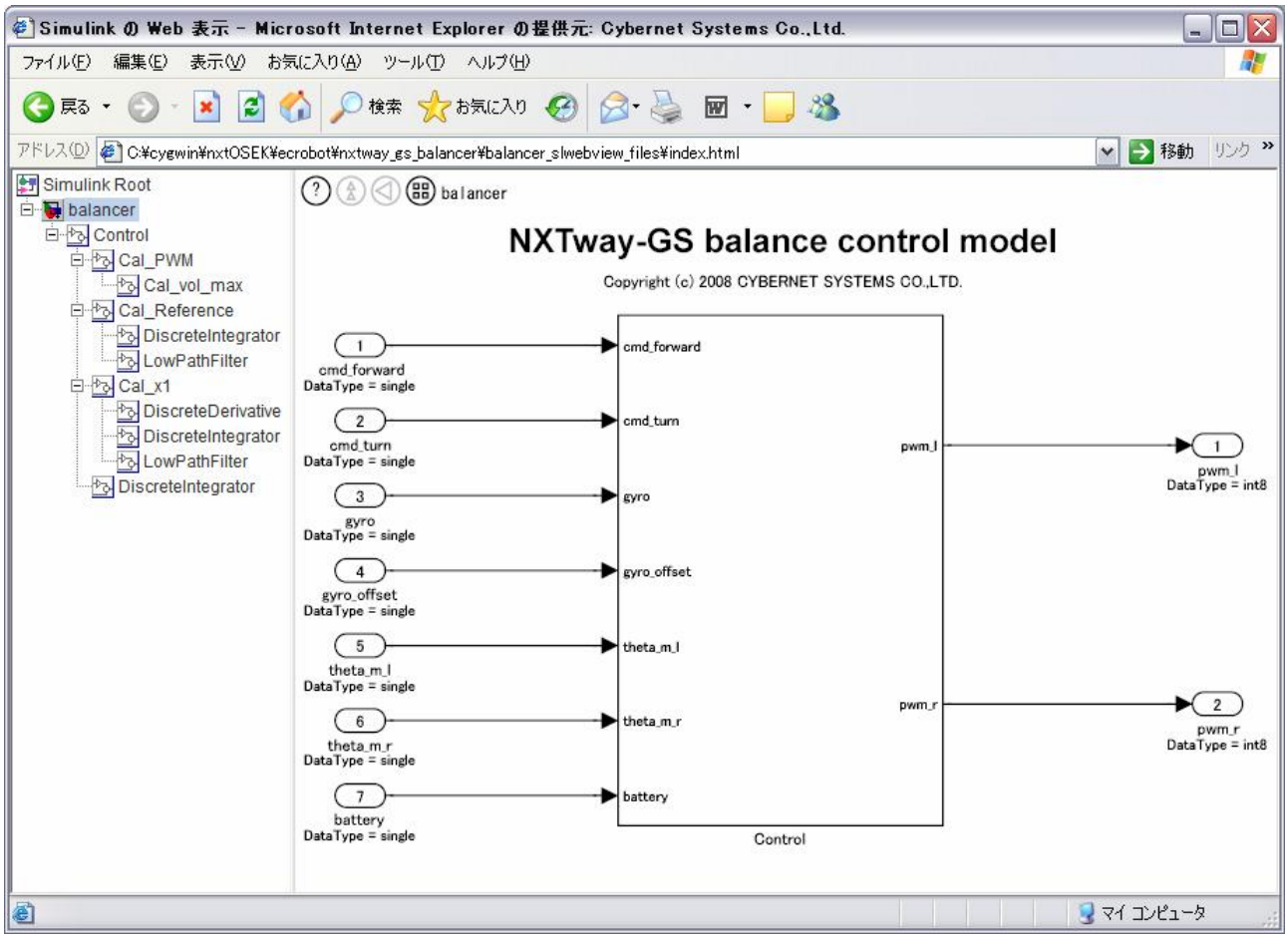
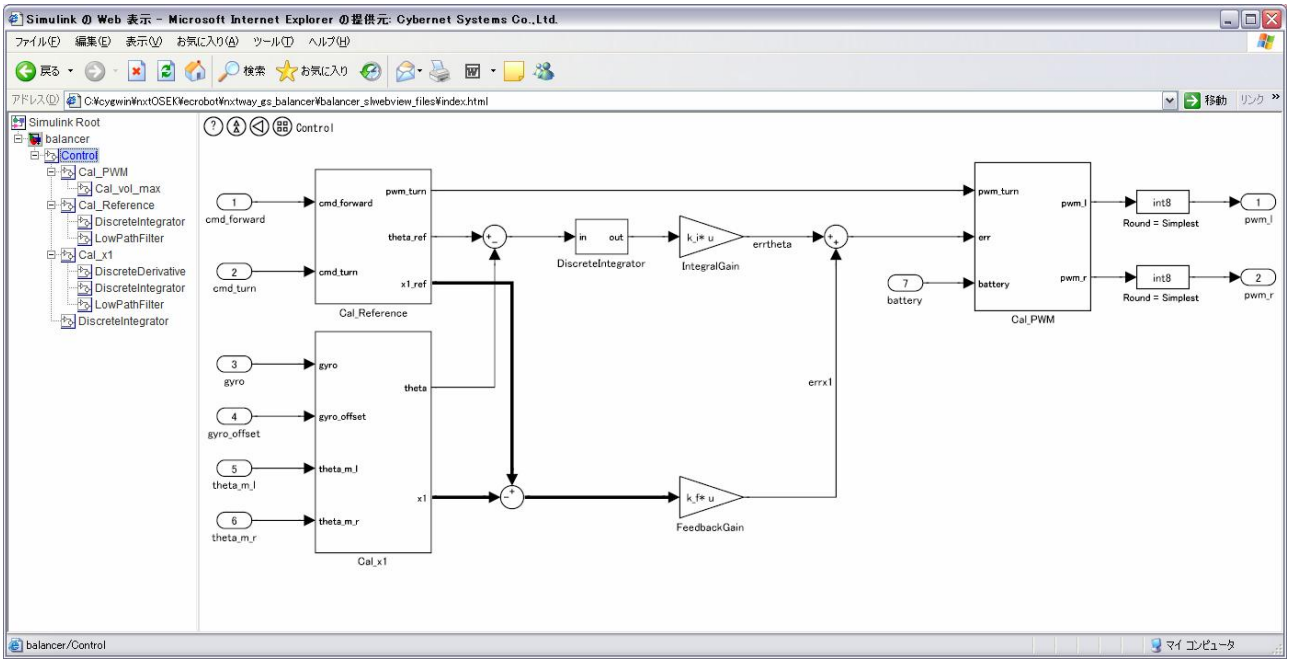



図 2-1 倒立振子制御 C API 用 Simulink モデル

Simulink では処理の固まりを Subsystem(サブシステム)と呼びます。図 2-2 はモデルの中核となる Control サブシステムの内部です。Cal_Reference サブシステムは倒立振子制御の目標値の演算部となります。Cal_x1 サブシステムは倒立振子の状態量(車輪平均回転角度、車体傾斜角度、車輪平均回転角速度、車体傾斜角速度)の演算部となります。また、Cal_PWM サブシステムは左右のモータ駆動 PWM 出力の演算部となります。

モデル中央にある三角形のブロックはゲインブロックと呼ばれるもので乗算処理をおこないます。中央上部の k_i*u と表記されたブロックは制御パラメータ K_I の乗算処理部になります。また中央下部の k_f*u と表記されたゲインブロックは制御パラメータ K_F の乗算処理部になります。信号線が太線の箇所は非スカラーデータの流れを示しています。Simulink におけるモデリングの特徴の一つとして、データをサイズに関わらず一本の信号線として表現できることが挙げられます。



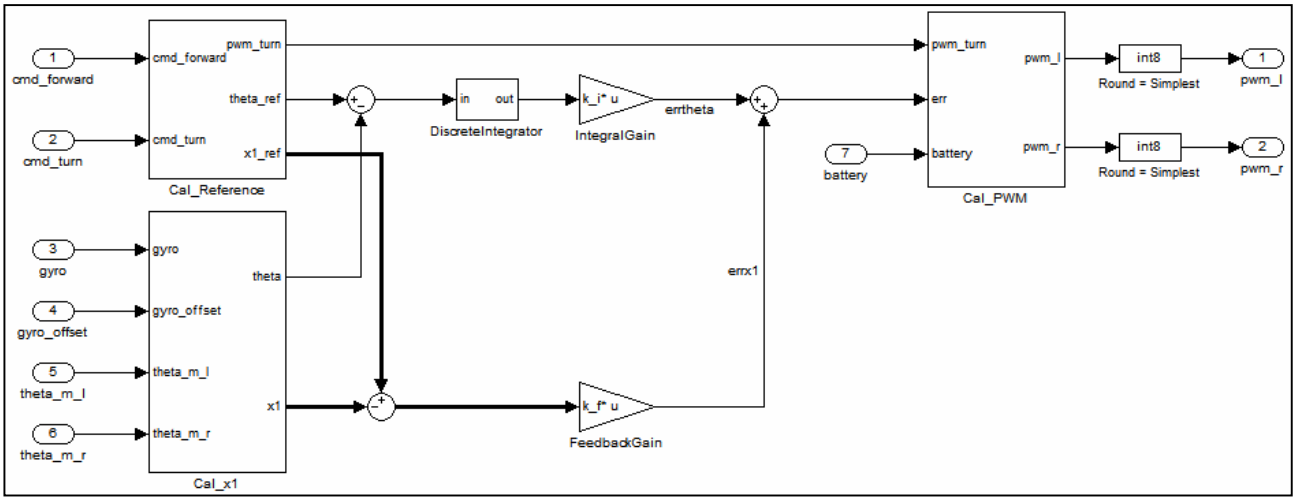




図 2-2 Control サブシステム

図 2-3 は、前後進命令/旋回命令に基づいた制御目標値を算出する制御目標値演算処理サブシステム(Cal_Reference)です。k_thetadot と表記されたゲインブロックは制御パラメータ K_THETADOT に対応した乗算部分です。倒立振子制御 C API では目標値として、車輪平均回転角度、車体傾斜角度(= 0)、車輪平均回転角速度、車体傾斜角速度(= 0)に対する 4 つの目標値を持っています。
また、k_phidot と表記されたゲインブロックは制御パラメータ K_PHIDOT に対応した乗算部分です。

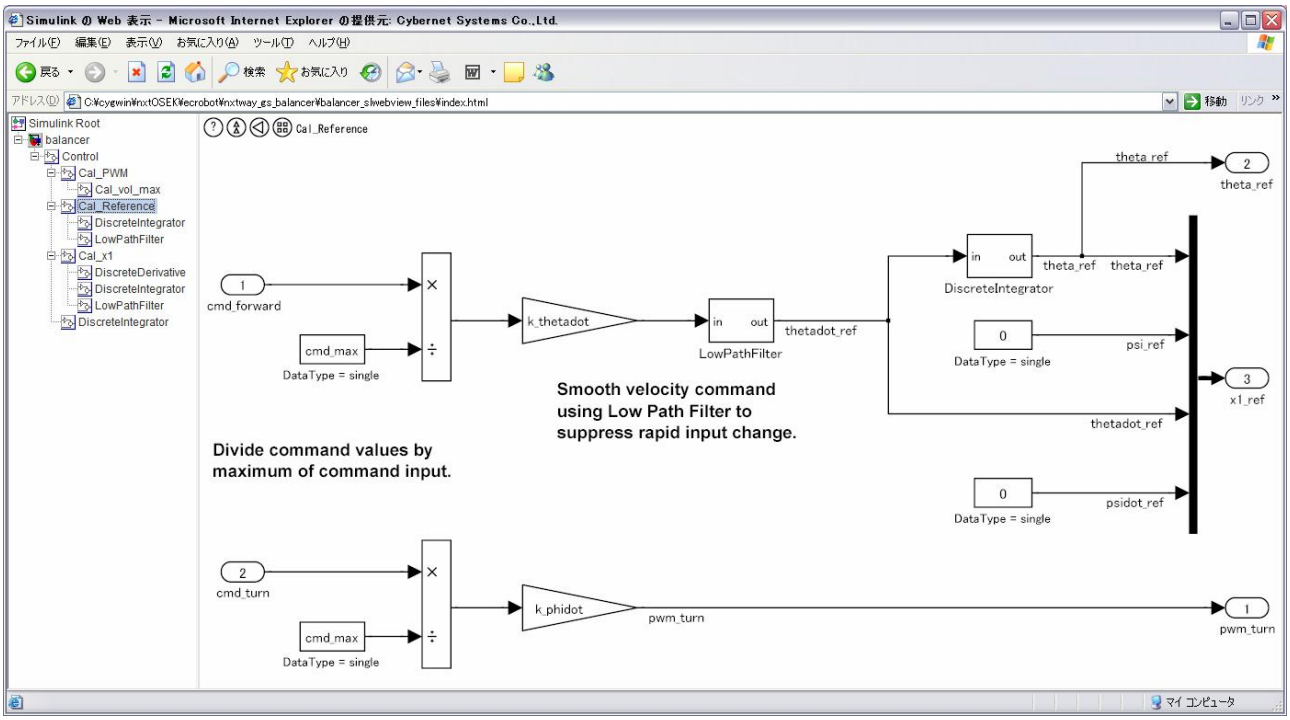



図 2-3 制御目標値演算処理サブシステム(Cal_Reference)内部

図 2-4 は、現代制御の状態方程式に基づいた倒立振子制御の状態量(車輪平均回転角度、車体傾斜角度、車輪平均回転角速度、車体傾斜角速度)を算出する状態量演算処理サブシステム(Cal_x1)です。

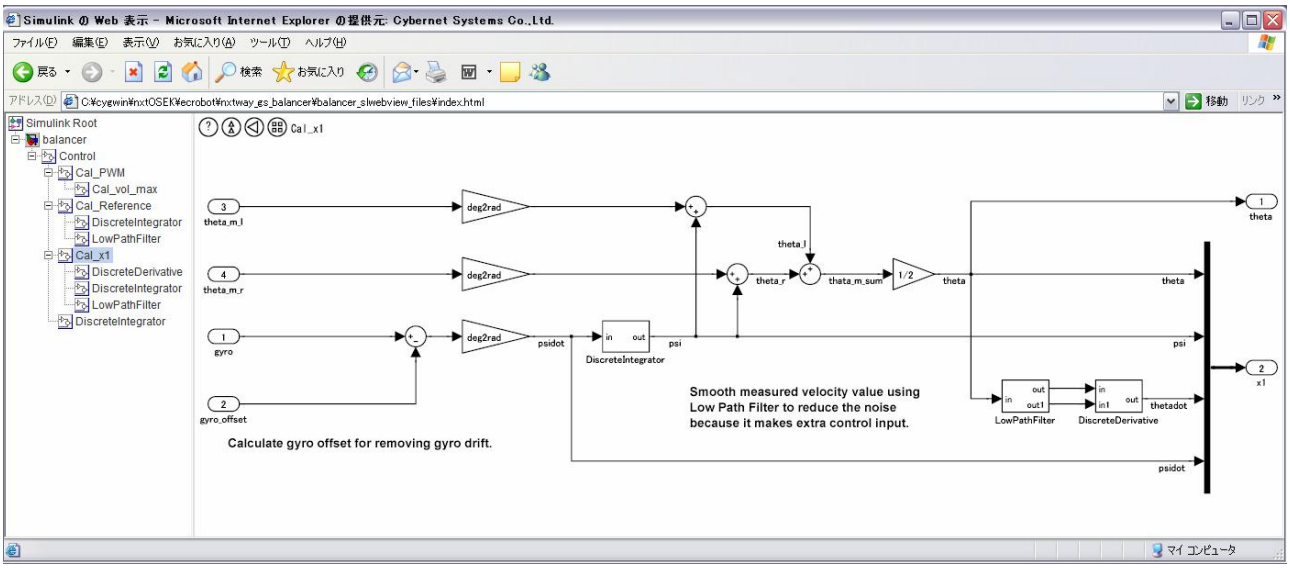



図 2-4 状態量演算処理サブシステム(Cal_x1)内部

図 2-5 は左右車輪の目標平均回転角度用ローパスフィルタ処理部分です。a_r と表記された部分が、左右車輪の目標平均回転角度用ローパスフィルタ係数 A_R に対応します。また 1/Z と表記されたブロックは UnitDelay ブロックと呼ばれ、前回演算値をホールドする機能を提供しています。
図 2-6 は左右車輪の平均回転角度用ローパスフィルタ処理部分であり、ローパスフィルタ係数 a_d (制御パラメータ A_D に対応)を除けば、図 2-5 と同じ処理内容になっています。

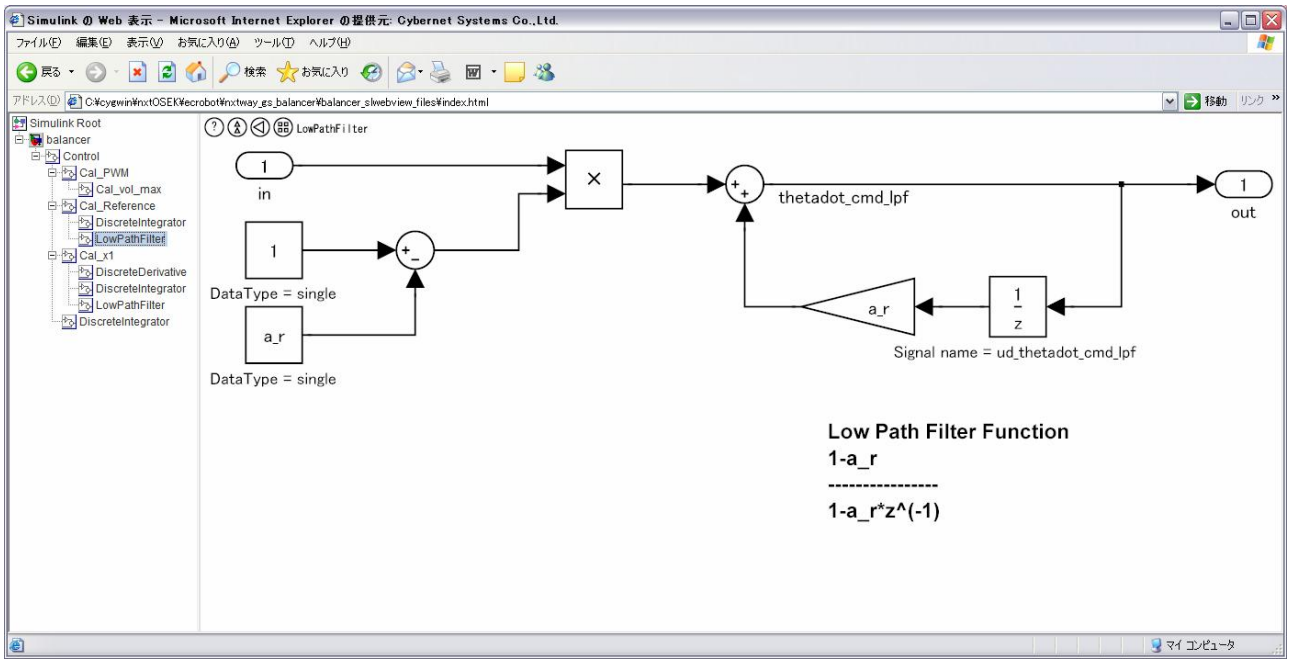



図 2-5 左右車輪の目標平均回転角度用ローパスフィルタ処理部

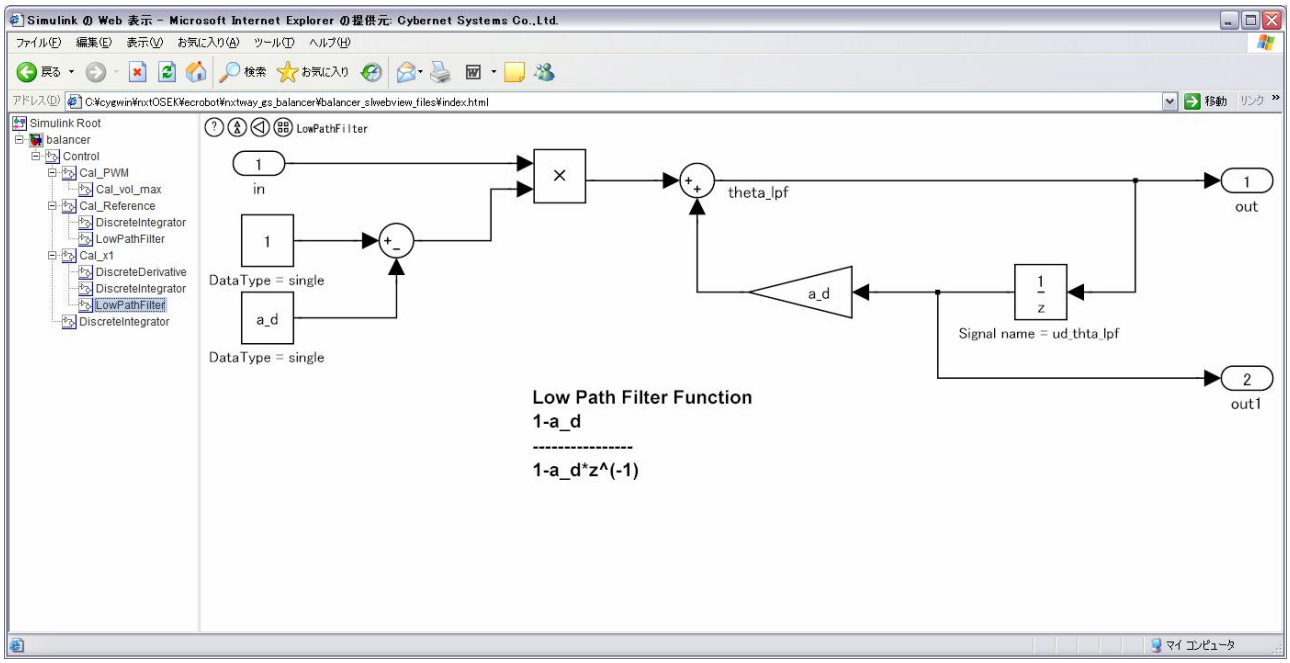



図 2-6 左右車輪の平均回転角度用ローパスフィルタ処理部

図 2-7 は Cal_PWM サブシステムの内部処理モデルです。NXTway-GS の旋回制御は pwm_turn からのデータを左右車輪モータ PWM 出力に±方向にオフセットさせることで実現しています。

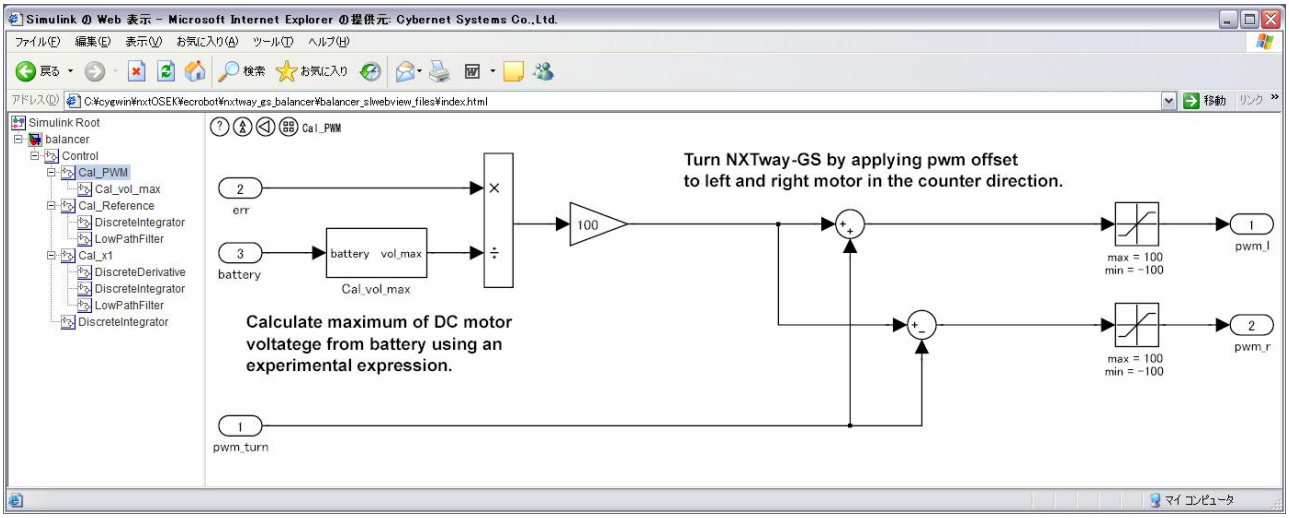



図 2-7 Cal_PWM サブシステムの内部処理

注記:
ユーザー独自の旋回制御をおこなうには、倒立振子制御の実行関数(balance_control)の第 2 引数を 0 に設定し(pwm_turn = 0)、戻り値の左右 PWM 出力に対して、独自の旋回制御に基づいた±方向の PWM 出力オフセットをそれぞれ与えることで実現できます。ここで重要なのは、旋回制御用 PWM 出力オフセットの絶対値は等しくするということです。これは倒立振子の前後方向の制御に必要な力と旋回方向の制御に必要な力のベクトルを直交させることで、2 つの制御間の干渉を防ぐためです。

図 2-8 はモータ駆動用 PWM 出力のバッテリ電圧補正演算サブシステム(CAL_vol_max)内部になります。battery_gain、battery_offset はそれぞれ制御パラメータ BATTERY_GAIN、BATTERY_OFFSET に対応しており、

バッテリ電圧補正後の PWM 出力ベース値 = BATTERY_GAIN×バッテリ電圧 - BATTERY_OFFSET

という一次式で補正されています。

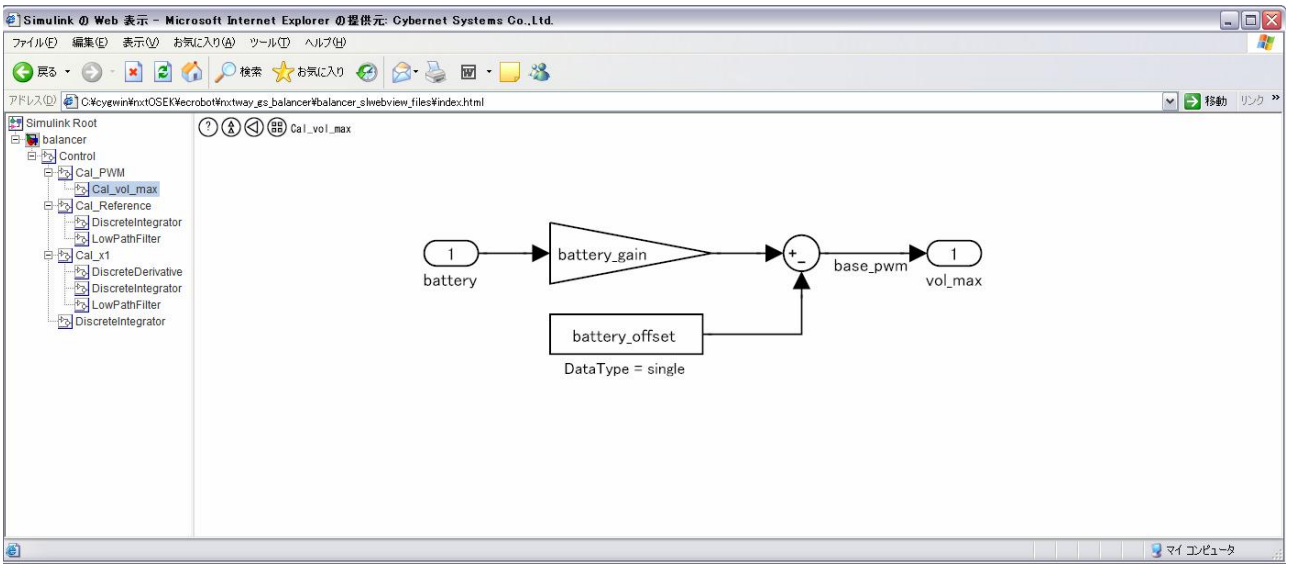



図 2-8 PWM 出力のバッテリ電圧補正処理部

# 3 サンプルプログラム

本章では、簡単なサンプルプログラムを用いて、NXTway-GS 倒立振子制御 C API の使い方について説明します。本サンプルプログラムを使用するには、nxtOSEK v2.07 以降の開発環境構築を完了している必要があります。本サンプルプログラムにおけるセンサ/モータの設定は次の通りです。

- タッチセンサ: ポート 1
- ジャイロセンサ: ポート 4
- 左車輪モータ: ポート C
- 右車輪モータ: ポート B

サンプルプログラムの起動後、ロボットを立たせた状態でタッチセンサを押すことで、倒立振子制御を開始します。

## 3.1 sample.c

sample.c はサンプルプログラムのメインソースファイルです。sample.c ファイルは TOPPERS/ATK(OSEK)用に記述されています。GYRO_OFFSET マクロは、センサ個体差に合わせて、適宜調整する必要があります。

nxtOSEK における TOPPERS/ATK の使用の際に、周期起動タスクを実装するには、user_1ms_isr_type2 という 1msec 周期起動の割り込みフック関数を利用しますが、サンプルプログラムでは、周期起動タスクは使用していないため、空の関数として実装されています。

OSEK_Task_bg タスクはバックグラウンドタスクであり、C 言語の main 関数と同様に、一度起動したら終了しないように、無限ループ内に繰返し処理を記述しています。倒立振子制御の初期関数(balance_init)を実行する際には、併せて左右の車輪駆動用モータエンコーダ値のゼロリセットをおこなう必要があります。倒立振子制御の実行関数(balance_control)は 4msec 周期起動を前提に設計されているため、サンプルプログラムでは無限ループの終端に 4msec のウエイト(systick_wait_ms(4))を配置しています。

```
/**
 ******************************************************************************
 **   ファイル名 : sample.c
 **
 **   概要       : NXTway-GS(2輪倒立振子ロボット)のTOPPERS/ATK(OSEK)用サンプルプログラム
 **
 ******************************************************************************
 **/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"
#include "balancer.h" /* 倒立振子制御用ヘッダファイル */

#define GYRO_OFFSET  610 /* ジャイロセンサオフセット値(角速度0[deg/sec]時) */

//*****************************************************************************
// 関数名  : user_1ms_isr_type2
// 引数    : なし
// 戻り値  : なし
// 概要    : 1msec周期割り込みフック関数(OSEK ISR type2カテゴリ)
//*****************************************************************************
void user_1ms_isr_type2(void){/* 空 */}

//*****************************************************************************
// タスク名: OSEK_Task_bg
// 概要    : バックグラウンド(常駐)タスク
//*****************************************************************************
TASK(OSEK_Task_bg)
{
      signed char pwm_L, pwm_R; /* 左右モータPWM出力 */

      while (ecrobot_get_touch_sensor(NXT_PORT_S1) == 0); /* タッチセンサ押下待機 */

      balance_init(); /* 倒立振子制御初期化 */
      nxt_motor_set_count(NXT_PORT_C, 0); /* 左モータエンコーダリセット */
      nxt_motor_set_count(NXT_PORT_B, 0); /* 右モータエンコーダリセット */

      while(1)
      {
        /* 倒立振子制御 */
        balance_control(
          (float)0, /* 前後進命令 */
          (float)0, /* 旋回命令 */
          (float)ecrobot_get_gyro_sensor(NXT_PORT_S4), /* ジャイロセンサ値 */
          (float)GYRO_OFFSET, /* ジャイロセンサオフセット値 */
          (float)nxt_motor_get_count(NXT_PORT_C), /* 左モータ回転角度[deg] */
          (float)nxt_motor_get_count(NXT_PORT_B), /* 右モータ回転角度[deg] */
          (float)ecrobot_get_battery_voltage(), /* バッテリ電圧[mV] */
          &pwm_L,  /* 左モータPWM出力値 */
          &pwm_R); /* 右モータPWM出力値 */
        nxt_motor_set_speed(NXT_PORT_C, pwm_L, 1); /* 左モータPWM出力セット */
        nxt_motor_set_speed(NXT_PORT_B, pwm_R, 1); /* 右モータPWM出力セット */

        systick_wait_ms(4); /* 4msecウエイト */
      }
}
```

## 3.2 balancer_param.c

balancer_param.c は倒立振子制御用パラメータの定義ファイルです。BATTERY_GAIN および BATTERY_OFFSET は const 定数として実装されているため、プログラム実行中は変更できませんが、その他のパラメータは初期値付き変数として実装されているため、プログラム実行中に変更することができます。

```
/**
 ******************************************************************************
 **    ファイル名 : balancer_param.c
 **
 **    概要       : 倒立振子制御パラメータ
 **
 ** 注記       : 倒立振子制御パラメータは制御特性に大きな影響を与えます。
 **
 ******************************************************************************
 **/

/*============================================================================
 * データ定義
 *============================================================================*/
float A_D = 0.8F; /* ローパスフィルタ係数(左右車輪の平均回転角度用) */
float A_R = 0.996F; /* ローパスフィルタ係数(左右車輪の目標平均回転角度用) */

/* 状態フィードバック係数
 * K_F[0]: 車輪回転角度係数
 * K_F[1]: 車体傾斜角度係数
 * K_F[2]: 車輪回転角速度係数
 * K_F[3]: 車体傾斜角速度係数
 */
float K_F[4] = {-0.870303F, -31.9978F, -1.1566F, -2.78873F};
float K_I = -0.44721F; /* サーボ制御用積分フィードバック係数 */

float K_PHIDOT = 25.0F; /* 車体目標旋回角速度係数 */
float K_THETADOT = 7.5F; /* モータ目標回転角速度係数 */

const float BATTERY_GAIN = 0.001089F; /* PWM出力算出用バッテリ電圧補正係数 */
const float BATTERY_OFFSET = 0.625F; /* PWM出力算出用バッテリ電圧補正オフセット */
```

## 3.3 sample.oil

OSEK では、OIL(OSEK Implementation Language)という設定ファイルに OS の設定を静的に定義する必要があります。

```
/**
 ******************************************************************************
 **   ファイル名 : sample.oil
 **
 **   概要       : サンプル用OSEK OIL(OSEK Implementation Language)ファイル
 **
 ** ※OSEK OIL記述方法詳細については
 **   nxtOSEK¥toppers_osek¥doc¥TOPPERS_OSEKカーネルSG取扱説書.pdfを参照してください。
 **
 ******************************************************************************
 **/
#include "implementation.oil"

CPU ATMEL_AT91SAM7S256
{
  OS LEJOS_OSEK /* nxtOSEKの旧名 */
  {
    STATUS = EXTENDED;
    STARTUPHOOK = FALSE;
    ERRORHOOK = FALSE;
    SHUTDOWNHOOK = FALSE;
    PRETASKHOOK = FALSE;
    POSTTASKHOOK = FALSE;
    USEGETSERVICEID = FALSE;
    USEPARAMETERACCESS = FALSE;
    USERESSCHEDULER = FALSE;
  };

  APPMODE appmode1{};

  /* OSEK_Task_bgタスク設定 */
  TASK OSEK_Task_bg
  {
    AUTOSTART = TRUE /* StartOSで自動的にREADY */
    {
      APPMODE = appmode1;
    };
    PRIORITY = 1;    /* 最低優先度 */
    ACTIVATION = 1;
    SCHEDULE = FULL;
    STACKSIZE = 512; /* bytes */
  };
};
```

## 3.4 Makefile

倒立振子制御 C API はライブラリとして提供されており、プログラムビルド時には Makefile 中にライブラリおよびインクルードパスを追加する必要があります。

```
# 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 = sample

# ライブラリの定義
USER_LIB = nxtway_gs_balancer
# インクルードパスの定義
USER_INC_PATH = $(NXTOSEK_ROOT)/ecrobot/nxtway_gs_balancer

# C ソースファイル
TARGET_SOURCES = balancer_param.c sample.c

# TOPPERS/ATK(OSEK)設定ファイル
TOPPERS_OSEK_OIL_SOURCE = sample.oil

# 下記のマクロは変更しないでください
O_PATH ?= build

# nxtOSEK ビルド用 Makefile のインクルード
include $(NXTOSEK_ROOT)/ecrobot/ecrobot.mak
```

## ■ 改訂履歴

| バージョン | 年月 | 改訂内容 | 著者・編者 |
|---|---|---|---|
| 1.0 | 2009/03 | 初版作成 | 近政 隆 |
| 1.1 | 2009/03 | 誤記訂正 | 近政 隆 |
| 1.2 | 2009/03 | 2.3 章のタイトルを変更他 | 近政 隆 |

本資料の内容・記載 URL は予告無く変更される場合があります。