Wii リモコンプログラミング

2011 年 9 月
（2014 年 5 月一部改訂）
徳島大学工学部知能情報工学科
光原弘幸

目次

１．Wii リモコンとは

Wii リモコンとは，Wii の標準コントローラのことで，海外では“Wiiremote”や“Wiimote”と呼ばれます（**図１．１**）．従来のゲームコントローラに見られる十字ボタン，A ボタンや B ボタンだけでなく，3 軸加速度センサ（モーションセンサ）や赤外線センサ等を搭載しており，コントローラを振ったり，傾けたり，画面に向けたりしてゲームを操作することができます．さらに，バイブレータで振動を発生させたり，内蔵のスピーカ（モノラル）から簡単なサウンドを流すこともできます．

このように，従来のゲームコントローラと比べて，入力及び出力インタフェイスが多様化されたことで，ゲーム内容の多様化も進んでいます．例えば，スポーツゲーム“Wii Sports”では，Wii リモコンをラケットやバットに見立てて，テニスや野球をプレイできます．また，ロールプレイングゲーム“DRAGON QUEST SWORDS”では，Wii リモコンを剣に見立てて，プレイヤが Wii リモコンを振るとキャラクタがモンスターに対して剣を振ることで，新感覚の戦闘シーンを提供しています．

Wii リモコンは Wii の標準コントローラですが“専用”コントローラではありません．Bluetooth（短距離無線通信技術）を介して PC にも接続可能で，ライブラリを用いて Wii リモコンプログラミングができます．ライブラリはいくつかあり，Windows や Mac だけでなく，Linux に対応するものもあります．ソフトウェア実験でも，Wii リモコンの特長を駆使して，新しさと面白さを兼ね備えたゲームの開発が期待されます．

この資料では，Wii リモコンプログラミングのごく基本的な部分のみを説明します．詳細は，Web やヘッダファイルを調べて確認してください．

図１．１　Wii リモコン

http://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:RVL-A-CW.jpg

２．Wii リモコンプログラミングの準備

２．１　Bluetooth 接続の確立

Wii リモコンは Bluetooth で PC と接続されます．よって，Wii リモコンプログラミングのためにはまず，Bluetooth 接続を確立することが必要です．幸い，電算室の PC には Bluetooth インタフェイス（アダプタ）が内蔵されています．

ここでは，Wii リモコンが Bluetooth 接続されているか確認をして，Wii リモコンの識別 ID を取得する方法を紹介します．

i. 手元に Wii リモコンを用意し，ターミナルを起動させます．

ii. 次のコマンドを入力し，hcitool を起動します．hcitool は，Bluetooth の接続を準備したり，接続状況を確認したりする機能を提供しています．

```
$ hcitool scan
```

iii. コマンドを入力したらすぐに，Wii リモコンの①と②ボタンを同時に押します．すると，Wii リモコンの LED が点滅するはずです（Wii リモコンから電波が出力されます）．

iv. hcitool が Wii リモコンからの電波を受信すると，ターミナルに次のような情報を提示します．

```
$ hcitool scan
Scanning ...
        00:22:D7:AA:14:D9       Nintendo RVL-CNT-01
```

この例では，00:22:D7:AA:14:D9 が Wii リモコンの識別 ID（アドレス）となります．得られた識別 ID を用いて，プログラムを作成したり起動したりすることになります．

２．２　Wii リモコンプログラミングのライブラリ

Wii リモコンプログラミングにはライブラリが必要です．いくつかのライブラリが提供されており，例えば，libwiiremote（libcwiimote），wiiuse，cwiid，WiiRemote Framework(DarwiinRemote)，aka.wiiremote，libwiimote，WiiYourself!（Native C++ Wiimote library）が代表的なものです．Linux 上で C 言語から Wii リモコンを扱うためには，libwiiremote，cwwid や libwiimote を利用します．

- cwiid：　http://abstrakraft.org/cwiid/
- libwiimote：　http://libwiimote.sourceforge.net/
- libwiiremote：　http://libwiiremote.sourceforge.jp/

電算室の PC には，　libwiimote（libcwiimote）ライブラリがインストール済みです．

自宅等の個人 PC に Wii リモコンプログラミング環境を構築する場合は，上記で示したライブラリをインストールしてください．個人の Linux PC が apt-get によるインストールに対応している場合，次のようなコマンドを入力することで，ライブラリのダウンロード・インストールが実行されます．

（1） cwiid ライブラリのインストール

```
$ sudo apt-get install libcwiid1-dev
```

- cwiid をインストールすると，libwiimote もインストールされるようです（バージョン古い？）．

（2） libwiimote

```
sudo apt-get install libcwiimote-dev
```

- ちなみに，make が成功すると，lib/libcwiimote.a，lib/libcwiimote.so が生成されます．

（3） libwiiremote

上記の Web サイトから，libwiiremote ライブラリをダウンロードできます．ただ，make をしようとすると，エラーが発生してしまいます．「BlueZ をインストールしていること」という前提が記されていますが，BlueZ をインストールして，Web 上にあるインストール対応策（例えば，http://www.hohog.net/node/74）に沿って試してみましたが，私の PC 環境では状況は打開されませんでした．古いバージョンの libwiiremote-0.1.6（http://sourceforge.jp/projects/libwiiremote/releases/）で試すと，ある程度インストールが進みますが，結局失敗に終わりました･･･

というわけで，申し訳ないですが，個人 PC への wiiremote ライブラリのインストールは，自分で試行錯誤してみてください．

SDL で Wii リモコンを扱えるライブラリがあると，理想的なのですが，Linux 版はまだ見つけるに至っていません（※見つけたら，知らせてください！）

３．Wii リモコンプログラミング

ここでは，SDL と libcwiimote を連携させた簡単なサンプルプログラム（一部）を示しながら，Wii リモコンプログラミングを説明していきます．と言っても，SDL と libcwiimote は独立しており，C 言語でそれぞれを呼び出しているに過ぎません．

なお，libcwiimote は cwiid の下位バージョンのようですが，他のライブラリと比べてプログラムできる Wii リモコン機能はさほど変わらず，比較的扱いやすいこともあり，ここでは libcwiimote を扱います（実際のところ，昨年度の課題でも用いた，という理由も大きいです）．最新版である cwiid を使いたい人は，http://abstrakraft.org/cwiid/wiki/libcwiid を参考にしてみてください．

３．１　事始め

（１）libcwiimote の利用を宣言

libcwiimote ライブラリを利用するには，まず，ソースコード内で libcwiimote に関するヘッダファイルをインクルードします．これにより，libcwiimote の関数や定数，グローバル変数などの定義がプリプロセッサとしてソースコードに埋め込まれます．関数の使用法などを確認したい時は，ヘッダファイル（/usr/include/ libcwiimote にあるはず）の中身を見てください．

インクルード文の例

```
#include <libcwiimote/wiimote.h>
#include <libcwiimote/wiimote_api.h>
```

（２）Wii リモコン変数を宣言

Wii リモコンに関する情報を格納する wiimote_t 型構造体を宣言します．これと同時に，定数 WIIMOTE_INIT で初期化します．

```
wiimote_t wiimote = WIIMOTE_INIT;
```

wiimote_t 構造体に，Wii リモコンの状態（情報）が格納されることになります．構造体の構成は次のようになります．構造体メンバについては，具体的な処理の説明の中で触れます．

```
typedef struct {
	wiimote_mode_t mode;  // 機能の状態（モード）
	wiimote_keys_t keys;  // キー（ボタン）の状態
	wiimote_point3_t axis;		//  各軸（3 軸）の加速度
	wiimote_ir_t ir1;	// 最初に検知した赤外線源
	wiimote_ir_t ir2;	// 2 番目に検知した赤外線源
	wiimote_ir_t ir3;	// 3 番目
	wiimote_ir_t ir4;	// 4 番目
	wiimote_cal_t cal;	// Wii リモコンのキャリブレーション
	wiimote_ext_port_t ext;		// 拡張ポートの状態
	wiimote_link_t link;  // リンク（接続）状態
	wiimote_led_t led;	//  LED の状態
	uint8_t rumble;		// 振動の状態
	uint8_t speaker;	// スピーカの状態
	uint8_t battery;	// バッテリの状態

	#ifdef _ENABLE_TILT
		wiimote_float3_t tilt;		// 各軸の Wii リモコンの傾き
	#endif
	#ifdef _ENABLE_FORCE
		wiimote_float3_t force;		// 各軸の重力加速度
	#endif

	struct {
		wiimote_mode_t mode;
		wiimote_keys_t keys;
		wiimote_led_t led;
		uint8_t rumble;
	} old;
} __attribute__((packed)) wiimote_t;
```

- wiimote_point3_t axis，wiimote_float3_t ，wiimote_float3_t は構造体であり，3 という数字が示すように，x，y，z の 3 軸に対応する変数から構成されます．例えば，tilt.x，tilt.y，tilt.z があります．
- #ifdef macro～#endif は，macro（マクロ）が定義されていた場合にのみ，コンパイル時に採用されます．マクロ定義は，gcc の-D コンパイルオプションで行います．マクロ定義がない場合は，コンパイルが通らないことがあります．

（3）Wii リモコンの接続

Wii リモコンは無線接続ですので，ゲームを開始するにはまず，接続を確立するための処理が必要になります．まずは，Wii リモコン識別 ID をコマンドライン引数としてプログラム（main 関数）に渡すことができるようにしましょう．

```
int main(int argc, char* argv[]) {
	if (argc < 2) {	// Wii リモコン識別 ID が与えられなければ
		printf("Designate the wiimote ID to the application.¥n");
		exit(1);
	}
}
```

- argc には，コマンドとそれに対する引数を足した数が渡されます．Wii リモコンに接続するには，その識別 ID が必須ですので，argc の値は 2 以上になるはずです．
- argc の値を見て，Wii リモコンの識別 ID が渡されているならば（そう想定できれば），次に wiimote_connect 関数による接続処理に移ります．

- **wiimote_connect 関数**（#include <wiimote_link.h>）
  Wii リモコンに接続します．
  *int wiimote_connect(wiimote_t *wiimote, const char *host);*
  引数（第 1 から）: 対象となる Wii リモコン（構造体アドレス），接続する Wii リモコンの識別 ID

```
if (wiimote_connect(&wiimote, argv[1]) < 0) {
	exit(1);
}
```

- 接続が成功すると，Wii リモコンの状態（情報）が渡した wiimote_t 型構造体に格納されます．
- 返値として，接続成功の場合は 0 を，接続失敗の場合は-1 を返します．
- *wiimote_t *wiimote_connect(bdaddr_t *bdaddr, wiimote_mesg_callback_t *mesg_callback, int *id);* という用法もあるようです．

複数の Wii リモコンに接続したい時は，argv[1]，argv[2]，argv[3]，･･･を指定します．例えば，次のようにすれば，複数の Wii リモコンに接続できるでしょう．

```
wiimote_t wiimote[4];
for (i=1; i < argc; i++){
	wiimote_connect(&wiimote[i], argv[i]);
}
```

コマンドライン引数を用いずに Wii リモコンを接続する方法もあります．その際，wiimote_discover 関数を用います．私の PC 環境では，この方法（以下の方法）はうまく動作しませんでした．深く調べたわけではないので，この方法で接続したい人は各自で調べてください．

- **wiimote_discover 関数**（#include <wiimote_link.h>）
  接続可能な Wii リモコンに探します．
  *int wiimote_discover(wiimote_t *devices, uint8_t size);*
  引数（第 1 から）：対象となる Wii リモコン（構造体アドレス），最大接続数

```
int wiimote_num;
wiimote_num = wiimote_discover(wiimote, 4);
if (wiimote_num == 0) {
        exit(1);
}
```

  - 返値として，接続可能な Wii リモコンがある場合はその台数を，ないの場合は 0 を返します．

そして，探し出せた Wii リモコンの台数分，接続処理を行います．

```
for (i=0; i<nmotes; i++) {
        if (wiimote_connect(&wiimote[i], wiimote[i].link.r_addr) < 0) {
                printf("unable to open wiimote: %s¥n", wiimote_get_error());
        }
}
```

  - wiimote[i].link.r_addr には，各 Wii リモコンの識別 ID が格納されます．l_addr もありますが，同じ Wii リモコンでも識別 ID が異なります．

（4）Wii リモコンの接続解除（切断）

Wii リモコンを接続して使い終わったら，wiimote_disconnect 関数で接続解除をしましょう．

- **wiimote_disconnect 関数**（#include <wiimote_link.h>）
  Wii リモコンの接続を解除します．
  *int wiimote_disconnect(wiimote_t *wiimote);*
  引数：対象となる Wii リモコン（構造体アドレス）

```
wiimote_disconnect(&wiimote);
```

（5）コンパイル

libwiimote を利用する場合，次のような gcc コンパイルオプションを指定します．これは，おまじない的に覚えておいてください

```
$ gcc wiil_test.c -lcwiimote -D_ENABLE_TILT -D_ENABLE_FORCE -L/usr/lib
```

  - 利用 PC 環境やライブラリのバージョンに依って，オプションが異なる場合があります．

３．２　イベント処理

Wii リモコンはキーボードやマウスと同じような入力デバイスであり（振動機構や LED，スピーカもあるので出力デバイスでもある），どのボタンが押されたか等のイベントに対応する処理を記述しなければなりません．となると，イベントドリブン・プログラミングの作法が適用されます．

では，Wii リモコンのイベント処理の流れを見ていきましょう．

３．２．１　イベントの取得方法

イベントの取得方法は，SDL におけるイベント取得とよく似ています．つまり，無限ループの中で，イベントを取得し，対応する処理を行います．若干異なるのは，明示的なイベントキューからイベント情報を取り出して処理すると言うよりは，どのボタンが押されたか等，Wii リモコンの状態を逐次的にに取得して対応する処理を行うというイメージでしょうか．

（1）ループ

イベント処理を行うために，「Wii リモコンがオープン（接続状態）であれば」という条件でループを回します．Wii リモコンがオープン（接続状態）かどうかを取得するには，wiimote_is_open 関数を用います．

- **wiimote_is_open 関数**（#include <wiimote_api.h>）

  Wii リモコンの接続を確認します．

  *int wiimote_is_open(wiimote_t *wiimote);*

  引数：対象となる Wii リモコン（構造体アドレス）

```
while (wiimote_is_open(&wiimote) { }
```

  - 返値として，接続状態であれば 1 を，そうでなければを 0（未接続）または-1（不明）を返します．
  - この関数の実体は，#define wiimote_is_open(w)　((w)->link.status == WIIMOTE_STATUS_CONNECTED)です．
  - 切断を確認する wiimote_is_closed 関数もあり，実体は!wiimote_is_open(w)になります．

（2）Wii リモコンの状態の取得・更新

イベント処理のための while ループの中で，イベントに対応する処理を行いますが，Wii リモコンの状態を頻繁に取得（処理する最新の値をプログラムに転送・反映させる）・更新（指定した値や処理を Wii リモコンに転送・反映させる）しなければ，即応性の高い（操作性の高い）ゲームにはなりません．

Wii リモコンの状態を取得・更新するには，wiimote_update 関数を用います．状態の取得・更新はできるだけ頻繁に行われるべきなので，while ループの最初に呼び出されるようにはしておきましょう．

- **wiimote_update 関数**（#include <wiimote_event.h>）

  Wii リモコンの状態を取得・更新します．

  *int wiimote_update(wiimote_t *wiimote);*

  引数：対象となる Wii リモコン（構造体アドレス）

```
while (wiimote_is_open(&wiimote)) {
	if (wiimote_update(&wiimote) < 0) {
		wiimote_disconnect(&wiimote);
		break;
	}
}
```

- 返値として，状態の取得・更新が成功したら 0 を，失敗ならばを-1 を返します.

（3）Wii リモコンのレポートタイプ

Wii リモコンから状態を取得する際，レポートタイプを指定することで，取得できる情報が異なってくるようです．例えば，ボタンの状態のみ取得するといった具合に，情報量を削減することで，Wii リモコンとの高速なやりとりが期待されます．ただ，正直，よく使い方が分からないので，ここでは関連する wiimote_report 関数を簡単に紹介するだけにします.

- **wiimote_report 関数**（#include <wiimote_report.h>）

  Wii リモコンのレポートタイプを指定する．

  *int wiimote_report(wiimote_t *wiimote, wiimote_report_t *report, uint8_t size);*

  引数：対象となる Wii リモコン（構造体アドレス），レポートタイプ，取得する状態情報サイズ

```
wiimote_report_t report = WIIMOTE_REPORT_INIT;
if (wiimote_report(&wiimote, &report, sizeof (report.status)) < 0) {
	wiimote_disconnect(&wiimote);
}
```

- wiimote_report_t 型構造体を宣言し，初期化してから関数を使うようです.

（4）モードとバッテリ残量

Wii リモコンの各種状態は．wiimote_t 構造体のメンバに格納されているので，その値に応じて処理することができます．例えば，Wii リモコンのモード（機能の状態）やバッテリ残量を表示するプログラムは次のようになります.

```
printf("MODE %02x¥n", wiimote.mode.bits);   // モード
printf("BAT %02x¥n", wiimote.battery);      // バッテリ残量
```

- メンバ mode.bits は，**表３．１**に示す値が格納されます．例えば，mode.acc，mode.ir ともに 1 が代入されている場合，モードは WIIMOTE_MODE_ACC_IR(0x33)になります.

表３．２　メンバ mode.bits に格納される定数とその値

定数（モード）	値
WIIMOTE_MODE_DEFAULT	0x30
WIIMOTE_MODE_ACC	0x31
WIIMOTE_MODE_IR	0x32
WIIMOTE_MODE_ACC_IR	0x33
WIIMOTE_MODE_EXT	0x34
WIIMOTE_MODE_ACC_EXT	0x35
WIIMOTE_MODE_IR_EXT	0x36
WIIMOTE_MODE_ACC_IR_EXT	0x37
WIIMOTE_MODE_FULL1	0x3e
WIIMOTE_MODE_FULL2	0x3f

３．２．２　ボタンイベントの種類と処理

Wii リモコンのボタン（キー）に関するイベントは，wiimote_t 型構造体の keys メンバにおける各ボタンに対応するメンバの値を見て進めます．値が 1 であればそのボタンが押されたことを示し，0 であれば押されていないことを示します．ボタンとメンバ及びボタン定数（マスク）の対応を**表３．２**に記載します．

例えば，次のように記述してイベントを処理します．

```
// A ボタンが押されたら
if (wiimote.keys.a) {
}
// A ボタンが離されたら
else {
}
```

wiimote_t.types のメンバの値を見る以外の方法もあります．それは，wiimote.keys.bits とボタン定数（ボタンマスク）との AND 演算の結果（値）を見る方法です．例えば，次のように記述します．

```
// A ボタンが押されたら
if (wiimote.keys.bits & WIIMOTE_KEY_A) { }
```

表３．２　Wii リモコンのボタンとメンバ及び定数の対応

ボタン	対応する wiimote_t.types のメンバ	ボタン定数（マスク）
１ボタン	wiimote_t.keys.one	WIIMOTE_KEY_1
２ボタン	wiimote_t.keys.two	WIIMOTE_KEY_2
A ボタン	wiimote_t.keys.a	WIIMOTE_KEY_A
B ボタン	wiimote_t.keys.b	WIIMOTE_KEY_B
十字ボタン上	wiimote_t.keys.up	WIIMOTE_KEY_UP
十字ボタン下	wiimote_t.keys.down	WIIMOTE_KEY_DOWN
十字ボタン右	wiimote_t.keys.right	WIIMOTE_KEY_RIGHT
十字ボタン左	wiimote_t.keys.left	WIIMOTE_KEY_LEFT
＋ボタン	wiimote_t.keys.plus	WIIMOTE_KEY_PLUS
－ボタン	wiimote_t.keys.minus	WIIMOTE_KEY_MINUS
HOME ボタン上	wiimote_t.keys.home	WIIMOTE_KEY_HOME
十字ボタン右上		WIIMOTE_KEY_UPRIGHT
十字ボタン左上		WIIMOTE_KEY_UPLEFT
十字ボタン右下		WIIMOTE_KEY_DOWNRIGHT
十字ボタン左下		WIIMOTE_KEY_DOWNLEFT

３．２．３　センサイベントの種類と処理

Wii リモコンの特徴（特長）の一つが，モーションセンサと赤外線センサです.

- モーションセンサ

3 軸（x, y, z）方向に各 8 ビットの加速度センサが内蔵されており，左右（X 軸），前後（Y 軸），上下（Z 軸）の加速度を取得できます.

- 赤外線センサ

赤外線 LED（Wii の周辺機器で言えばセンサバー）から発せられる赤外線を CMOS センサで検知することで，Wii リモコンの先端がどの方向に向いているか（ポインティングしているか）を取得できます.

これらのセンサを使用するにはまず，センサからのデータを受け付けるモードにするために，次のような記述が必要です.

```
wiimote.mode.acc = 1
```

- 値を 0 にすると，センサからのデータを受け付けません.
- acc は加速度センサの ON/OFF に関わると思われますが，赤外線センサ使用時も acc が 1 でないと動作しないようです（私の PC 環境だけ？）

（1）加速度

Wii リモコンでは 3 軸の加速度を取得できます．Libwiimote におけるリモコンと軸の関係を**図３．１**に示します．なお，Wii リモコンプログラミングに関する書籍等を見ると，軸の配置は図３． 1 と一致していますが，プラス方向とマイナス方向が反転しています．libwiimote の仕様で反転しているのでしょうか…

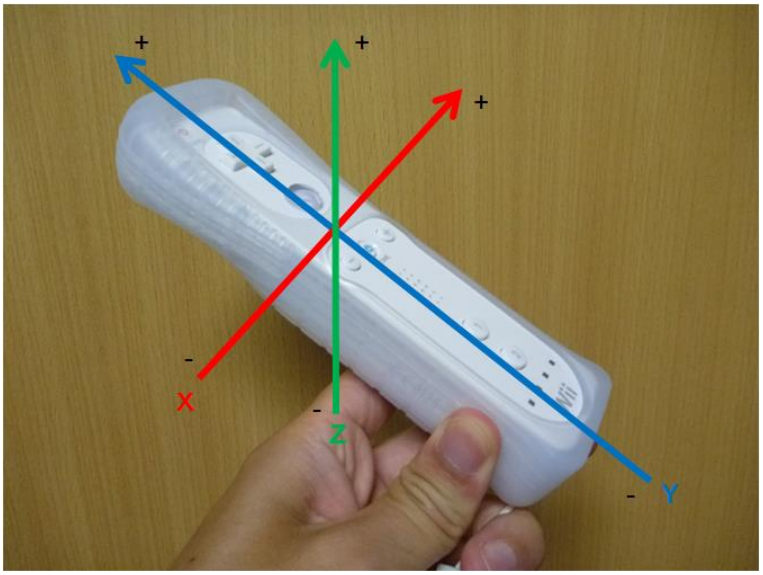

図３．１　Wii リモコンと座標軸

加速度の値は，wiimote_t 構造体メンバ axis.x，axis.y，axis.z に格納されますので，それらの（現在の）値を wiimote_update 関数で Wii リモコンから取得して処理します．例えば，加速度のターミナルへの表示は次のような記述になります．

```
printf("AXIS　x=%03d　y=%03d　z=%03d¥n",　wiimote.axis.x,　wiimote.axis.y,
wiimote.axis.z);
```

- 値は 0～255 までの 256 階調です．軸に対してプラス方向，マイナス方向がありますので，だいたい 127 を境に加速度の方向を判別できることになります．Wii リモコンを静止させると，そのような値になるはず（静止させたとしても，重力加速度の影響を受けて，１つの軸の値が増減します）．

加速度の値を参照することで，どの方向にどれくらいの力で Wii リモコンを振ったかが分かります．

（2）傾き

Wii リモコンの 3 軸に対する傾きを取得できます．傾きは，得られた加速度から算出することもできますが，libwiimote では wiimote_t 構造体メンバ tilt.x，tilt.y，tilt.z に格納されます．例えば，傾きのターミナルへの表示は次のような記述になります．

```
printf("TILT　x=%.3f　y=%.3f　z=%.3f¥n",　wiimote.tilt.x,　wiimote.tilt.y,
wiimote.tilt.z);
```

- 値は-90.0～90.0 までの float 型です．各軸に対して±90 度までの値に相当します．

（3）重力加速度

Wii リモコンの 3 軸に対する重力加速度を取得できます．傾きは，得られた加速度から算出することもできますが，libwiimote では wiimote_t 構造体メンバ force.x，force.y，force.z に格納されます．例えば，重力加速度のターミナルへの表示は次のような記述になります．

```
printf("FORCE　x=%.3f　y=%.3f　z=%.3f　(sum=%.3f)¥n",　wiimote.force.x,
wiimote.force.y, wiimote.force.z,
sqrt(wiimote.force.x*wiimote.force.x+wiimote.force.y*wiimote.force.y+wiimote.
force.z*wiimote.force.z));
```

（4）赤外線センサ

赤外線センサを用いるには，当たり前ですが，センサバー（赤外線 LED）をディスプレイの上部あたりに設置する必要があります．そして，プログラムでは，wiimote_t 構造体のメンバ mode.acc に 1 を代入にするとともに， mode.ir にも 1 を代入して，赤外線利用モードを指定します．そうすると，wiimote_t 型構造体のメンバ ir1.x, ir1.y, .ir1.size や ir2.x, ir2.y, .ir2.size に，Wii リモコンによりポインティングされた座標値が格納されます．

ir1 と ir2 の違いは，前者が赤外線を最初に（1 つ）検知した際の座標，後者が 2 番目（2 つ）検知した際の座標のようです．3 つ，4 つの赤外線にも対応していますが，センサバーの赤外線は 2 つなので，ir2 までにしか適切な座標値は格納されません．

例えば，ポインティング座標のターミナルへの表示は次のような記述になります．

```
printf("IR1　x=%04d　y=%04d　ss=%d¥n",　wiimote.ir1.x,　wiimote.ir1.y,
wiimote.ir1.size);
printf("IR2　x=%04d　y=%04d　ss=%d¥n",　wiimote.ir2.x,　wiimote.ir2.y,
wiimote.ir2.size);
```

ここで，実際にポインティング座標を表示させてみると，x 軸（プラスとマイナス）が反転していることが分かります．さらに，x，y 座標ともに，300，400 台の値を取得できていないようです．これは libwiimote の仕様なのか，センサバーの特性なのか，単にプログラムの洗練が足りていないのか…詳しい検証はできていないのですが，試行錯誤で適切な座標取得ができるように工夫してください．

例えば，x 軸の反転に対しては，次のように対処することで，プログラム（ディスプレイ）の座標系に合わせることができます．

```
x = abs(wiimote.ir2.x - window->w);
```

３．２．４　出力イベントの種類と処理

Wii リモコンは入力だけでなく，次のような出力インタフェイスも有しています.

- 振動

Wii リモコンには振動を発生させるバイブレータが 1 つ内蔵されており，ON/OFF を制御することができます.

- スピーカ（モノラル）

Wii リモコンには，モノラルスピーカが専用のサウンドプロセッサとともに内蔵されています．このプロセッサは高度なものではなく，ADPCM（Adaptive Differential Pulse Code Modulation）方式で 4 ビットのビープ音的なサウンドを指定して鳴らすことができるようです（完全には解析されていない？）．とはいえ，効果音としての機能が期待できます.

- LED

Wii リモコンの上面下部（Wii のロゴの上）に LED（青色）が 4 つ搭載されています．これら LED の点灯／消灯を制御できます．表現力が高いとは言えませんが，使い方は工夫次第で広がります.

（1）振動

Wii リモコンを振動させるには，wiimote_t 型構造体の runble メンバに 1 を代入します．振動を止めるには 0 を代入します.

```
wiimote.rumble = 1;
```

（2）スピーカ

Wii リモコンのスピーカから音を鳴らすのはいくつかの手順を踏みます．まず，鳴らす音のサンプリングデータを次のように配列に格納して与えます.「どのような値を与えると，どのような音が出るか」は各自で試してみてください.

```
uint8_t sample[20] = {
        0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,
        0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c,0x3c };
```

次に，wiimote_speaker_init 関数を用いてスピーカを初期化します.

- **wiimote_speaker_init 関数**（#include <wiimote_speaker.h>）

  Wii リモコンのスピーカを初期化します.

  *int wiimote_speaker_init(wiimote_t *wiimote, uint8_t fmt, uint8_t freq);*

  引数（第 1 から）: 対象となる Wii リモコン，フォーマット，周波数

```
wiimote_speaker_init(&wiimote, WIIMOTE_FMT_S4, WIIMOTE_FREQ_44800HZ);
```

  - fmt に指定できる定数は, WIIMOTE_FMT_S4 と WIIMOTE_FMT_S8 です. 前者は 0x00 に，後者は 0x40 が値になります（何のフォーマット？）.
  - freq に指定できる定数は，WIIMOTE_FREQ_44800HZ，WIIMOTE_FREQ_33600HZ，WIIMOTE_FREQ_22400HZ，WIIMOTE_FREQ_11200HZ です．0x0a，0x0e，0x16，0x1a が値になります（何の周波数？）.

そして，次に示す関数を用いて，出力周波数と音量を調整し，サンプリングデータを再生します.

- **wiimote_speaker_freq 関数**（#include <wiimote_speaker.h>）

  Wii リモコンのスピーカ出力周波数を指定します.

  *int wiimote_speaker_freq(wiimote_t *wiimote, uint8_t freq);*

  引数（第 1 から）: 対象となる Wii リモコン，周波数

```
wiimote_speaker_freq(&wiimote, 100)
```

  - freq に与える値と出力される音の関係は各自で試してみてください.

- **wiimote_speaker_volume 関数**（#include <wiimote_speaker.h>）

  Wii リモコンのスピーカの音量を指定します.

  *int wiimote_speaker_volume(wiimote_t *wiimote, uint8_t volume);*

  引数（第 1 から）: 対象となる Wii リモコン，音量（ボリューム）

```
wiimote_speaker_volume(&wiimote, 100);
```

  - volume の値を 0 にすると消音（ミュート）になります.
  - 消音する int wiimote_speaker_mute(wiimote_t *wiimote);もあります.

- **wiimote_speaker_play 関数**（#include <wiimote_speaker.h>）

  Wii リモコンのスピーカからサンプリングデータを再生します.

  *int wiimote_speaker_play(wiimote_t *wiimote, uint8_t *buf, uint32_t size);*

  引数（第 1 から）: 対象となる Wii リモコン，サンプリングデータ，サンプリングデータサイズ

```
wiimote_speaker_play(&wiimote, sample, 20);
```

  - size には，サンプリングデータの要素数を指定します.

Wii リモコンのスピーカを使い終えたら，wiimote_speaker_free 関数でスピーカ再生に要した資源を解放します.

- **wiimote_speaker_free 関数**（#include <wiimote_speaker.h>）

  Wii リモコンのスピーカを解放します.

  *int wiimote_speaker_free(wiimote_t *wiimote);*

  引数：対象となる Wii リモコン

```
wiimote_speaker_free(&wiimote);
```

（3）LED

LED を点灯／消灯させるには，wiimote_t 構造体のメンバ led.one，led.two，led.three，led.four に 1（点灯）または 0（消灯）を代入します．led.one は一番左の LED，led.four が一番右の LED を示します．一番左と一番右の LED を点灯させるには次のようにします.

```
wiimote.led.one  = 1;
wiimote.led.four  = 1;
```

このように, 各 LED に対して点灯／消灯を制御できる他, 次のように記述することとで 2 進数的に LED を制御できます.

```
wiimote.led.bits += 1;        // 光らせる LED を増やす（2 進数的に）
wiimote.led.bits -= 1;        // 光らせる LED を減らす（2 進数的に）
```

３．２．５　Wii リモコンの拡張

Wii リモコンには，ヌンチャクやジョイスティックといった拡張コントローラを接続可能です．拡張コントローラの制御は, wiimote_t 構造体のメンバ ext.nunchuk や ext.classic を参照することで行います．ここでは説明を割愛しますので，拡張コントローラを使う場合は，各自で調べてみてください.

４．おわりに

この資料では，wiimote ライブラリ（libcwiimote）を用いた Wii リモコンプログラミングについて簡単に説明しました. 情報量として十分ではないと思いますが，ULS に上げているサンプルプログラムも参考にしながら，試行錯誤でプログラミングしてみてください.

最後に，libwiimote に限らず Wii リモコンプログラミングの参考になるであろう，または興味関心を高めてくれるであろう URL を示します.

- http://libwiimote.sourcearchive.com/lines/0.2/main.html
- http://libwiiremote.sourceforge.jp/libwiiremote/doc/ja/api/index.html
- http://wiibrew.org/wiki/Wiimote/Library
- http://www.kako.com/neta/2006-019/2006-019.html
- http://www.hohog.net/node/74
- http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=978-4-274-06750-1
- http://www.kosaka-lab.com/tips/

５．おまけ

ここでは，SDL と wiiremote を連携させた簡単なサンプルプログラム（一部）を示しながら，Wii リモコンプログラミングを説明していきます..

３．１　事始め

（1）wiimote ライブラリの利用を宣言

wiimote ライブラリを利用するには，まず，ソースコード内で wiimote に関するヘッダファイルをインクルードします．これにより，wiimote の関数や定数，グローバル変数などの定義がプリプロセッサとしてソースコードに埋め込まれます．関数の使用法などを確認したい時は，ヘッダファイルの中身を見てください.

インクルード文の例

```
#include <wiiremote.h>
#include <wiiremote_utils.h>
```

- とりあえず，これらのヘッダファイルをインクルードしておきましょう.
- ヘッダファイルの依存関係は，wiiremote_utils.h⊂wiiremote.h⊂stdlib.h となります.
- wiimote ライブラリの変数，定数，関数は，“WRMT_”から始まります.

（2）Wii リモコン変数を宣言

Wii リモコンに関する情報を格納する WRMT_WiiRemote 型変数（ポインタ）を宣言します．Wii リモコンは複数台接続可能ですので，想定する台数分の情報を格納するために配列等を用います.

```
WRMT_WiiRemote *wiiremotes[WRMT_MAX_DEVICES];      // WRMT_MAX_DEVICES=32
```

（3）Wii リモコン接続処理と初期化

Wii リモコンは無線接続ですので，ゲームを開始するにはまず，接続を確立するための処理が必要になります．さらに，接続した Wii リモコンの初期化として，どの機能を使用するかも指定します.

複数の Wii リモコンに対する接続処理は while ループを用いて，例えば以下のように記述します.

```
int wii_remote_num, tmp_num;
// Wii リモコンの接続がない間は繰り返す（1 台でも接続すると処理は終了）
while (wii_remote_num <= 0) {
	// Wii リモコンライブラリの初期化
	if (WRMT_Init() != 0) {
		exit(-1);
	}
	tmp_num = WRMT_GetNumWiiRemote();	// // 接続台数の取得
	// Wii リモコンとの接続
	for (i = 0; i < tmp_num; i++) {
		WRMT_IOReturn rc;	// Wii リモコンの接続状況を格納する変数
		wiiremotes[i] = WRMT_GetWiiRemoteAt(i);	// インデックスを指
定し，Wii リモコンの情報を取得
		rc = WRMT_WiiRemote_Open(wiiremotes[i]);	// Wii リモコンに接続
		// 接続エラーの場合
		if (rc == WRMT_IO_ERROR) {
			exit(-1);
		}
		// Wii リモコンの使用する機能を有効化
		WRMT_WiiRemote_SetEnabled(wiiremotes[i],WRMT_FUNCTION_CONTINUO
	US, 1);
		WRMT_WiiRemote_SetEnabled(wiiremotes[i], WRMT_FUNCTION_MOTION,
1);
		WRMT_WiiRemote_SetEnabled(wiiremotes[i], WRMT_FUNCTION_IR, 1);
		WRMT_WiiRemote_Update(wiiremotes[i]);	// Wii リモコンの状態
更新
	}
	Wii_remote_num = tmp_num;	// 接続数を格納
}
```

- **WRMT_Init 関数**（#include <wiiremote.h>）

  Wiimote ライブラリを初期化します.

  int　WRMT_Init ()

```
if (WRMT_Init() != 0) { }
```

  - 返値が 0 であれば，初期化は成功です.

- **WRMT_GetNumWiiRemote 関数**（#include <wiiremote.h>）

  Wii リモコンの接続台数を取得します.

  int WRMT_GetNumWiiRemote ();

```
tmp_num = WRMT_GetNumWiiRemote();
```

- **WRMT_GetWiiRemoteAt 関数**（#include <wiiremote.h>）
  インデックスを指定して Wii リモコンの情報を取得します.
  *WRMT_WiiRemote * WRMT_GetWiiRemoteAt (int device_index);*
  引数：対象の Wii リモコンのインデックス番号

```
wiiremote = WRMT_GetWiiRemoteAt(0);
```

  - 返値として，情報の取得が成功した Wii リモコンの情報を返します..
- **WRMT_WiiRemote_Open 関数**（#include <wiiremote.h>）
  指定した Wii リモコンに接続します.
  *WRMT_IOReturn　WRMT_WiiRemote_Open (WRMT_WiiRemote *self);*
  引数：対象の Wii リモコン

```
WRMT_IOReturn rc;
rc = WRMT_WiiRemote_Open(wiiremote);
```

  - 返値として，指定した Wii リモコンの接続状況を返します. 接続状況を表す定数は以下の通りです.
    - WRMT_IO_TIMEOUT：接続タイムアウト（=-2）
    - WRMT_IO_ERROR ：接続失敗（=-1）
    - WRMT_IO_SUCCESS ：接続成功（=0）
- **WRMT_WiiRemote_SetEnabled 関数**（#include <wiiremote.h>）
  Wii リモコンで使用する機能を有効／無効化します.
  *void　WRMT_WiiRemote_SetEnabled　(WRMT_WiiRemote　*self,　WRMT_FunctionType function_type, int value);*
  引数（第 1 から）：対象の Wii リモコン，有効／無効化する機能，有効／無効

```
WRMT_WiiRemote_SetEnabled(wiiremote, WRMT_FUNCTION_CONTINUOUS, 1);
WRMT_WiiRemote_SetEnabled(wiiremote, WRMT_FUNCTION_MOTION, 0);
```

  - function_type には以下の定数を指定します.
    - WRMT_FUNCTION_CONTINUOUS ：
    - WRMT_FUNCTION_MOTION ：
    - WRMT_FUNCTION_IR：赤外線センサ（ポインティング用）
    - WRMT_FUNCTION_SPEAKER：スピーカ（Wii リモコン内蔵）
    - WRMT_NUMBER_OF_FUNCTIONS ：
  - value に 1 を指定すると有効，0 を指定すると無効になります.
- **WRMT_WiiRemote_Update 関数**（#include <wiiremote.h>）
  Wii リモコンの状態（情報）を更新します.
  *WRMT_IOReturn　WRMT_WiiRemote_Update (WRMT_WiiRemote *self);*
  引数：対象の Wii リモコン

```
WRMT_WiiRemote_Update(wiiremote);
```

- 返値として，更新成功の場合は WRMT_IO_SUCCESS を，失敗の場合は WRMT_IO_ERROR を返します．