平成23年度卒業論文

Bluetoothを用いたダンスロボットの遠隔操作

函館工業高等専門学校　情報工学科　5年
東海林研究室　松井秀樹

目次

1章　序論

1.1　はじめに

近年、産業用ロボット以外にも一般家庭向けロボットなどの様々なロボットが人間社会に入り込んできたことから、ロボットと人間社会の共生社会に関する研究・実験が盛んになっている。その中でも特に音楽・ダンス分野は人間とのコミュニケーションが直接的に関わってくる分野である。合奏、セッション、ダンスの相手やボーカル役にロボットを使うのは、ロボットと人の共生を追求する上で重要な課題である[1]。例えば、京都大学では人間の音楽表現に合わせて自身も楽器演奏を行うことのできる共演者ロボットが開発され[2]、産業技術総合研究所では人間と共に踊り、人間に極めて近い動作を可能にした「HRP-4C 未夢」が開発された[3]。

一方、函館高専門学校でも、ロボットを学校の様々な公報活動の場で使っているが、そのような公報活動の場でもより多くの人々に興味を持ってもらえるようなロボットを必要としている。

1.2　研究背景

本研究では、函館高専と公立はこだて未来大学が共同開発をしたイカロボット「IKABO」(図 1-1)を人間の楽器演奏に合わせて動作させるシステムの開発を目的とした。しかし想定したシステムを実際に構築するには、人員の確保、コスト面の問題がある。そこで目的を達成するための前段階として市販されているロボットを動作させるシステムを開発した。本システムの構成図を図 1-2 に示す。

図 1-1. イカロボット「IKABO」

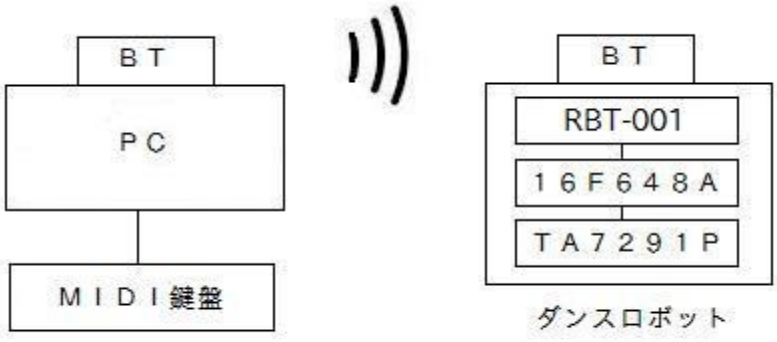

図1-２．　システム構成図

1.3　英文アブストラクト

We developed a system which makes a robot dance according to performance information
that has been sent from a musical keyboard. Recently various robots have joined in human society.
Hakodate national College of Technology has also been doing P.R. using a robot called "IKABO".
However, many operators and large space are required to operate "IKABO". Therefore, we developed a
remote system that can operate cheap commercial robots using Bluetooth. Additionally, users can
operate robots easily using a musical keyboard that is connected to the system.

2章　ロボット制御について

この章では、ロボット制御の概要、開発環境について説明する。

2.1　開発環境

ロボット制御 PIC の開発環境として OS に Windows Vista、開発プラットフォームに MPLAB IDE v8.06 を使用した[4]。また PIC ライターはAKI－PICプログラマー Ver.4を使用した[5]。

2.2　ロボット制御の概要

本研究ではロボット制御用 PIC として PIC16F648A[6][7]、モータドライバに TA7291P [8]、Bluetooth の送受信モジュールに 80FG990 で拡張した RBT-001[9]を使用している。ロボット制御の回路図を図 2-1 に示す。また組み立て後のロボットを図 2-2 に示す。PIC が PC から Bluetooth を介して送られてきた演奏情報に従い TA7291P を通じてモータの正転・逆転・停止、速度調整を行う。これらの制御を演奏情報が送られてくる間繰り返す。PC からの演奏情報の受信には RBT-001 を用いることにより RS232C[10]による通信が可能となっている。これらをブレッドボード上に配置・配線する。また、動作させるロボットは株式会社寺子屋製の「ダンシング猫田係長」(図 2-3)[11]を使用している。

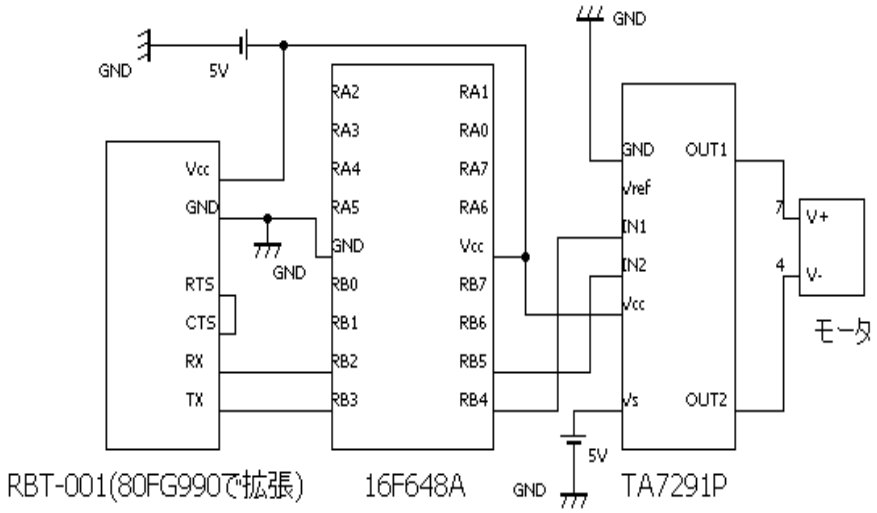

図 2-1. ロボット制御の回路図

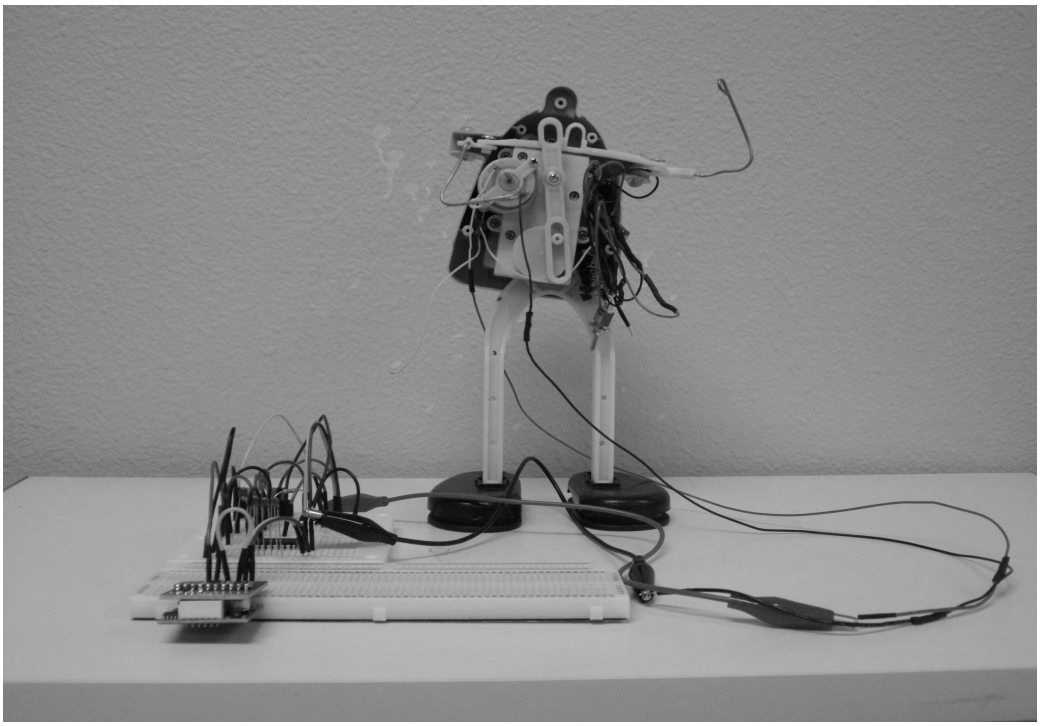

図 2-2. 組み立て後のロボット

図 2-3. 「ダンシング猫田係長」

2.3　RBT-001

RBT-001 は、29×29(mm)という小さい基板内にアンテナを搭載した Bluetooth 通信規格の組込用モジュールである[9]。本体にアンテナを搭載しており通信距離は約 30m まで可能で、RS232C 通信を仮想化して Bluetooth 機器間で無線通信を行うことができる。UART 側のシリアル通信速度は最大 921.6kbps を実現している。

また、5V の電源電圧で RBT-001 用の 3V の電源を作って供給するために 80FG990 で拡張している。RBT-001 のロジック信号の電圧レベルは 3V なので 5V の回路とはそのまま直結できないが、80FG990 によって 5V を 3V の信号レベルに電圧調整できる。80FG990 で拡張した RBT-001 のピン配置と外観を表 2-1、図 2-4に示す。

表 2-1. 80FG990 で拡張した RBT-001 のピン配列

ピン番号	名称	方向	
1	TX	0	URAT - 送信データ
2	RX	1	URAT - 受信データ
3	CTS	1	URAT - 送信可能(アクティブ Low)
4	RTS	0	URAT - 受信可能(アクティブ Low)
5			何も接続しない
6	GND	-	電源 GND
7	Vcc	1	電源電圧+5 ～ +6

※方向は 0 は出力、1は入力を表している。

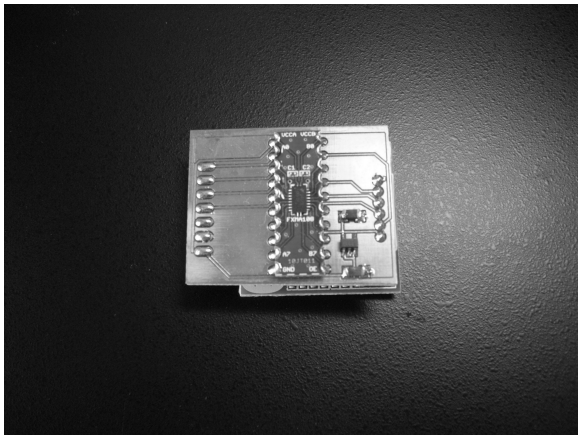

図 2-4. 80FG990 で拡張した RBT-001 の外観

2.4 MPLAB IDE

MPLAB IDE(Integrated Development Environment)は Windows ベースの PIC 用統合開発環境を提供するフリーソフトウェアである[4]。ここではソースを作成し PIC に書きこむまでの使用方法について順を追って記述する。

（１）プロジェクトを作成する。

①ツールバーの「Project」→「Project Wizard...」をマウスでクリックする。
②ウイザード画面が表示されるが、そのまま[次へ(N)>]をクリックする。
③ Step One ではデバイスの選択をする。”Device”の[▼]をクリックして作成したいPICの種類を選択し、[次へ(N)>]をクリックする。
④ Step Two ではコンパイラの選択をする。”Active toolsuite”の[▼]をクリックして「CCS C Compiler .....」を選択し、[次へ(N)>]をクリックする。
⑤ Step Three ではプロジェクトの保存先の選択をする。[Browse...]をクリックし、フォルダを選択し、ファイル名を入力して[保存(S)]をクリックする。その後ウィザードが出てきたら[次へ(N)>]をクリックし、[完了]で終了する。

(2)プログラムソースを作成する。

①ツールバーの「New File」をクリックして[Untitled]のエディター画面を表示する。
②このエディター画面にソースを書き込む。
③ツールバーの「Save File」をクリックし、ファイル名を入力し、"Add File To Project"にチェックを入れる。その後「保存(S)」をクリックする。

(3)コンパイルをする。

①ツールバーの「Project」→「Build All」をクリックしてコンパイルをする。
②[OutPut]ウインドウ画面が出てくるので、エラーが出ていたらソースを訂正し再びコンパイルをする。

(4)PIC ライターをストレートケーブルで接続して PIC をセットする。

(5)PIC に書き込む。

① Windows スタートメニューから「PIC プログラマ」を起動する。
②「通信」ボタンをクリックし、PIC ライターのポートを選択する
③「HEX ロード」ボタンをクリックし、作成した HEX ファイルをロードする。
④「プログラム(P)」ボタンをクリックすると PIC にプログラムが書き込まれる。

2.5　作成したプログラム

作成したプログラムを付録(P.16)に示す。PC から”a”を受信するとモータを停止する。”b”を受信するとモータが正転し、”c”ならばモータが逆転する。”1”を受信するとモータの速度が速くなり、”2”ならば遅くなる。”3”を受信すると”1”と”2”の中間速度になる。

3章　演奏情報送信側プログラムについて

この章では、PC 側の演奏情報送信側プログラムの概要、開発環境について説明する。

3.1　開発環境

演奏情報送信プログラムの開発環境として OS に UbuntuLinux 11.04、開発言語 に C++言語を使用した。なお、Bluetooth 通信用のライブラリは BlueZ[12][13]を使用している。また MIDI メッセージの取得のためにライブラリの ALSA[14]を使用した。

3.2　演奏情報送信プログラムの概要

PC 側の処理の流れは以下の通りとなる。なお今回は楽器として MIDI 鍵盤を用いた。

(1)RS232C 通信を行うために RFCOMM ポートを使用し仮想シリアルポートを作成する[15]。
(2)PC に接続した MIDI 鍵盤から MIDI メッセージを取得する。
(3)取得した MIDI メッセージからモータの正転・逆転・停止、速度の決定を行う。
(4)その後 Bluetooth を介して演奏情報を送信する。
(5)(2)～(4)を繰り返す。

3.3　RBT-001 との接続

PC と RBT-001 を Bluetooth で接続する手順を以下に示す。

(1)PC の Bluetooth デバイスと RBT-001 を起動する。
① PC の USB デバイスに Bluetooth デバイスを接続する。今回は BT-MicroEDR2X を使用した。
②設計した回路に電源を接続し RBT-001 の電源が入ったことを確認する (RBT-001 の LED が緑に光る)。

(2)PC 側でペアリングを行う。
① Bluetooth の「設定」から「新規デバイスの追加」をクリックする。
②ウィザードが出てくるので「進む(F)」をクリックする。
③しばらくするとデバイスが一覧に出てくるので「EasyBT」を選択する。
④ペアリングが成功すると「設定」画面にデバイスが表示され、コンセントマークが表示される。

(3)仮想シリアルポートを作成する。

① PC で端末を起動する。

②”hcitool scan”コマンドで接続先のデバイスとアドレスを確認する。

③”sudo rfcomm connect 0 アドレス”で RBT-001 と接続する。接続に成功すると”dev/rfcomm0 to アドレス on channel 1”と表示される。

④以上の実行例を図 3-1 に示す。

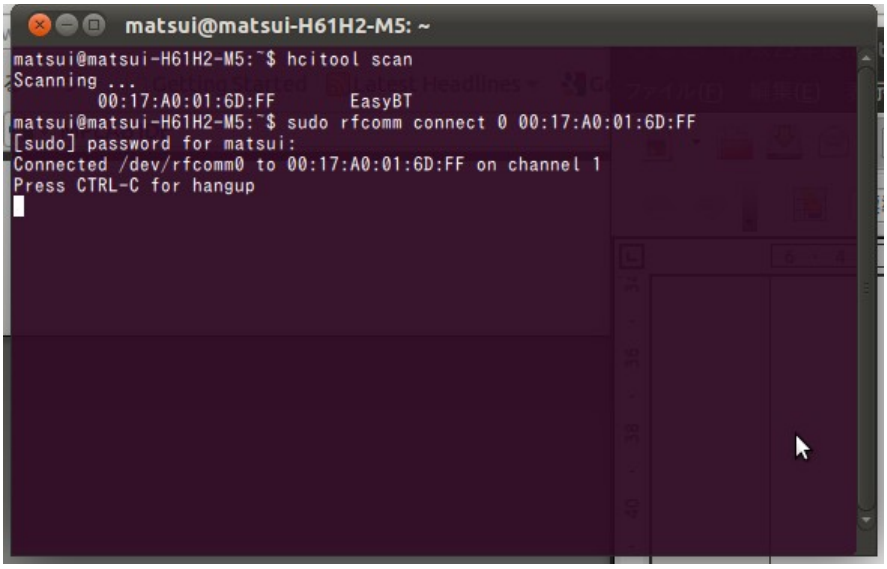

図 3-1. 仮想シリアルポート作成例

(4)別端末を開きプログラムを実行する。

3.4　作成したプログラム

作成したプログラムを付録として添える(P.18)。プログラムを実行すると、MIDI 鍵盤から演奏情報が送られてくるのを待つ。演奏情報が送られてくると MIDI メッセージに対応したモータの回転方向と速度の決定をする。その後仮想シリアルポートを介して RBT-001 にデータを送信する。

4章　動作実験と結果

この章では、動作実験と結果について説明する。

4.1　実験概要

実際にPC側のプログラムを起動し、MIDI鍵盤で演奏して動作を確認した。テスト項目を表4-1に示す。

表4-1　テスト項目

実験項目
(1)PCとRBT-001がペアリングできているか。
(2)Bluetoothで正常にデータを送信できているか。
(3)送られてきたデータを正常に受信できているか。
(4)受信データをもとに対応した動作をしているか。
(5)MIDI鍵盤に対して遅延が発生していないか。
(6)速く演奏をしても正常に動作をするか。

4.2　実験結果

各項目について(1)から(5)までは正常な動作を確認したが、(6)については速くMIDI鍵盤を押しすぎると送受信バッファがすぐに溜まり正常な動作をしないという問題が発生した。

4.3　考察

今回作成したシステムでは基本的な動作はしたが、演奏速度が速いと送受信バッファがすぐに溜まり正常な動作をしなくなる問題が生じた。実際の公報活動の場では楽器による曲演奏に合わせて動作をさせるので、ハード側、PC側両方のプログラムの改善が求められる。

また、ハードウェアの回路をロボットの中に内蔵する予定だったが予想以上に回路が大きくなってしまったので縮小する必要がある。

また、動作がぎこちないので自然にダンスをしているように動きを調整をする必要がある。例えば、今回作成したシステムでは演奏情報としてMIDIメッセージを使用しているが、MIDI鍵盤を押す強さを示す値(ノート・ベロシティ)を速度調整の制御に利用する。

また、PC を介さずに楽器にマイコンを搭載し、直接ロボットに演奏情報を送信することでシステムの全体の縮小化を目指す。

さらに、現在のシステムでは MIDI 鍵盤を用いているが、今後はギター、ドラムなど様々な楽器から演奏情報を送信し動作をさせる必要がある。

５章　まとめ

本研究では、Bluetooth を介して楽器からの演奏情報をロボットへ送信し、その演奏情報に合わせてロボットを動作をさせるシステムの開発を行った。

なお、このシステムを平成 24 年 3 月 10 日に本校で実施される「ものつくり成果の体験・展示会」に展示し、被験者からアンケートを取って改善点を見つける予定である。

参考文献

[1]音楽ロボットのための実時間音楽情報処理, 奥乃博, 中臺一博, 大塚琢馬, 情報処理, Vol.50, No.8 (2009).

[2]共演者音楽ロボットのためのパーティクルフィルタを用いた実時間楽譜追従手法, 大塚琢馬, 情報処理, Vol.53, No.12 (2011).

[3]Wikipedia HRP-4C,
http://ja.wikipedia.org/wiki/HRP-4C

[4]MPLAB IDE,
www.microchip.com/

[5]AKI-PIC プログラマー
http://akizukidenshi.com/catalog/g/gK-02018/

[6] 8 ピン PIC マイコンの使い方がよくわかるかる本, 後閑哲也, 株式会社技術評論 (2010).

[7] C 言語による PIC プログラミング入門, 後閑哲也, 株式会社具術表鵜論社出版 (2002).

[8]モータドライバ IC を使う,
http://www9.plala.or.jp/fsson/NewHP_elc/H8/H8_53mterDrv.html

[9]RBT-001,
http://park11.wakwak.com/~microtechnica/cgi-bin/goodslist.cgi?mode=view_detail&genre_id=00000028&goods_id=00000001

[10]PIC16F648A のシリアル通信機能,
http://it.mech.hi-tech.ac.jp/~ono/pic16f648a/pic16f648a-UART.html

[11]ダンシングキャット,
http://www.zakkaya-aguna.com/12_161.html

[12]BlueZ,
http://www.bluez.org/

[13]Bluetooth programing in C with BlueZ,
http://people.csail.mit.edu/albert/bluez-intro/c404.html

[14]ALSA - wikipedia,
http://ja.wikipedia.org/wiki/Advanced_Linux_Sound_Architecture

[15]Ubuntu/Bluetooth シリアルポート,
http://debugitos.main.jp/index.php?Ubuntu%2FBluetooth%A5%B7%A5%EA%A5%A2%A5%EB%A5%DD%A1%BC%A5%C8

付録

ロボット制御のプログラム

```
//基本的なプログラムの順番
//PIC の設定を読み込む
#include <16F648A.h>
//PIC の動作モードを設定
#fuses INTRC_IO, NOLVP, NOWDT, PUT, NOPROTECT, NOBROWNOUT, NOMCLR //CCP1B3
//内蔵クロックは、最大 4MH zなので、最大に。
#use delay (clock=4000000)
//シリアル通信のときは fast_io()が必須です。
#use fast_io(B)

//シリアル通信の設定。
#use rs232(baud=9600,xmit=PIN_B2,rcv=PIN_B1)
void get_data();

///// メイン /////
void main()
{
 int i;
 int  high = 32,low = 0, ra;
 int cmd1; //command data

 //各ピンのインプット、アウトプットの設定
 //0・・出力に。1・・入力に。
 set_TRIS_A(0b00000100);

 //シリアル受信ピンに指定したピン（ここでは PIN_B)を、入力に。
 set_TRIS_B(0b00000010);

 //最初に全部消してリセットします。しなくても OK。
 output_A(0b00000000);

 //起動確認　10回点滅
 for (i=0; i<10; i++){
  output_A(0b11111111); delay_ms(50);
  output_A(0b00000000); delay_ms(50);
```

```
 }
 //メインループ
 printf("test¥r¥n");
 while (true){
 //ここにPICにずっとやってもらいたいことを書く

 /// get command from PC
 if(kbhit()) //PIC に受信データが届いたかどうか
 {
  printf("kbhit¥r¥n");
  cmd1 = getchar(); //コマンド受信
  printf("getc¥r¥n");
  switch (cmd1)
   {
   case 'a': ra = (0x00);
    break;

   case 'b': ra = (0x01);
                break;
   case 'c': ra = (0x02);
                break;
   case '1': high = 120;          //早くなる
          low  = 128 - high;
          break;
   case '2': high = 70;           //遅くなる
          low  = 128 - high;
          break;
   case '3': high = 90;
          low =  128 - high;      //中間
          break;
   default: break;
  }
 }
 output_a(ra);
 delay_ms(high);
 output_a(0);
 delay_ms(low);
 }
}
```

演奏情報送信プログラム

```
// コンパイル  g++ sound.cpp -lasound
// amidi -l で MIDI デバイスを検出して main 関数の sound.play( "hw:～" ); の内容を書き換える

#include "sound.h"
#include "serial.h"

#include <stdio.h>

//serial
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>

//mid
#include <signal.h>
#include <alsa/asoundlib.h>
#include <unistd.h>
#include <pthread.h>

#include <assert.h>
#include <string.h>

#define _DEBUG_MSG

enum
{
   STATUS_COMMAND = 0,
   STATUS_NOTE,
   STATUS_VELOCITY
};


////////////////////////////////////////////////
//メイン


int main()
{
   Sound sound;
   sound.play( "hw:1,0,0" );

   getchar();
```

```
    sound.stop();
    return 0;
}



////////////////////////////////////////////////
// Sound

// コンストラクタ
Sound::Sound()
{
#ifdef _DEBUG_MSG
    printf("Sound¥n");
#endif

    serial.init();                                          //serial の初期化を追加

    thread = 0;
    stop_thread = false;
    key = 0;
    speed = 0;

    memset( note_on, 0, sizeof( note_on ) );
}


// デストラクタ
Sound::~Sound()
{
    stop();
}


// 停止
void Sound::stop()
{
    if( is_playing() ){
        stop_thread = true;
        while( is_playing() ) usleep( 10 );
    }
}


// スレッドを起動して再生
void Sound::play( const char* device )
{
    if( is_playing() ) return;
```

```
    stop_thread = false;
    if( device ) midi_device = device;

    pthread_create( &thread, NULL, Sound::launcher, (void *) this );
    pthread_detach( thread );
}


// スレッドのランチャ
void* Sound::launcher( void* args )
{
    Sound* sound = ( Sound * ) args;
    sound->play_thread( );
    return NULL;
}



// 再生スレッド本体
void Sound::play_thread()
{
    int ret;

    // MIDI デバイスオープン
    snd_rawmidi_t *dev_midi = NULL;
    if( ! midi_device.empty() ){

        const int mode = SND_RAWMIDI_NONBLOCK;
        ret = snd_rawmidi_open( &dev_midi, NULL, midi_device.c_str(), mode );
        if( ret < 0) {
            fprintf( stderr, "snd_rawmidi_open 失敗 %s: %s", midi_device.c_str(), snd_strerror( ret ) );
            thread = 0;
            return;
        }
    }

    // その他変数初期化
    int status = STATUS_COMMAND;
    int command = -1;
    int note = -1;
    unsigned char midi_message;

    // stop()を呼ぶまでループ
    while( ! stop_thread  ){

        // MIDI 入力
        ret = snd_rawmidi_read( dev_midi, &midi_message ,1 );

        if( ret != -EAGAIN ){
```

```
        // ステータスバイト取得
        if( status == STATUS_COMMAND ){

#ifdef _DEBUG_MSG
            if( midi_message != 0xFE &&
                midi_message!=0xF8 &&
                midi_message != 0x00 )
                fprintf(stderr,"----------¥nmessage = %02x¥n", midi_message );
#endif

            // ノートオン
            if( midi_message == 0x90 ){
#ifdef _DEBUG_MSG
                printf( "note on¥n");
#endif
                command = midi_message;
                status = STATUS_NOTE;
            }

            // ノートオフ
            else if( midi_message == 0x80 ){
#ifdef _DEBUG_MSG
                printf( "note off¥n");
#endif
                command = midi_message;
                status = STATUS_NOTE;
            }

            // データバイトの場合はランニングステータス
            else if( midi_message & 0x7f ){

                for( int i=1; i< MAX_SOUNDS; i++ ){

                    if( note_on[i] == midi_message ){
                        note = midi_message;
                        status = STATUS_VELOCITY;
                        break;
                    }
                }
            }
        }

        // ノートナンバー取得
        else if( status == STATUS_NOTE ){
            note = midi_message;
            status = STATUS_VELOCITY;
#ifdef _DEBUG_MSG
            printf( "note = 0x%02x¥n", note);
```

```
#endif
        }

        // ベロシティ取得
        else if( status == STATUS_VELOCITY ){

            // 再生
            if( command == 0x90 && midi_message > 0){

                for( int i = 1; i < MAX_SOUNDS; i++ ){

                    if( midi_play( note, i ) ){
#ifdef _DEBUG_MSG
                        printf("play¥n" );
                        for( int j = 1; j < MAX_SOUNDS; ++j ) printf("note[ %d ] = 0x%02x(%d)¥n",j,
note_on[j], note_on[j]);
#endif
                        break;
                    }
                }
            }

            // 停止
            else{

                for( int i = 1; i< MAX_SOUNDS; i++ ){

                    if( midi_stop( note, i ) ){
#ifdef _DEBUG_MSG
                        printf("stop¥n" );
                        for( int j = 1; j < MAX_SOUNDS; ++j ) printf("note[ %d ] = 0x%02x(%d)¥n",j,
note_on[j], note_on[j]);
#endif
                        break;
                    }
                }
            }

            status = STATUS_COMMAND;
        }
    }

  }// while( ! stop_thread  )


  // デバイスクローズ
  if( dev_midi ){
    snd_rawmidi_drain( dev_midi );
    snd_rawmidi_close( dev_midi );
```

```
    }

    thread = 0;

#ifdef _DEBUG_MSG
    printf("sound closed.¥n");
#endif
}


// 鍵盤押した
const bool Sound::midi_play( unsigned char note, const int sound_num )
{
  if( note_on[ sound_num ] ) return false;
  note_on[ sound_num ] = note;

  int hz;
  switch( note ){

      case 33 : hz = 110;  break;
      case 34 : hz = 117;  break;
      case 35 : hz = 123;  break;

      case 36 : hz = 131;  key = 1;  speed = 1;  break; // O2C
      case 37 : hz = 139;  key = 2;  speed = 1;  break;
      case 38 : hz = 147;  key = 1;  speed = 2;  break;
      case 39 : hz = 156;  key = 2;  speed = 2;  break;
      case 40 : hz = 165;  key = 1;  speed = 3;  break;
      case 41 : hz = 175;  key = 1;  speed = 1;  break;
      case 42 : hz = 185;  key = 2;  speed = 3;  break;
      case 43 : hz = 196;  key = 1;  speed = 2;  break;
      case 44 : hz = 208;  key = 2;  speed = 1;  break;
      case 45 : hz = 220;  key = 1;  speed = 3;  break;
      case 46 : hz = 233;  key = 2;  speed = 2;  break;
      case 47 : hz = 247;  key = 1;  speed = 1;  break;

      case 48 : hz = 262;  key = 1;  speed = 2;  break; // O3C
      case 49 : hz = 277;  key = 2;  speed = 3;  break;
      case 50 : hz = 294;  key = 2;  speed = 3;  break;
      case 51 : hz = 311;  key = 2;  speed = 1;  break;
      case 52 : hz = 330;  key = 1;  speed = 1;  break;
      case 53 : hz = 349;  key = 2;  speed = 2;  break;
      case 54 : hz = 370;  key = 2;  speed = 2;  break;
      case 55 : hz = 392;  key = 1;  speed = 3;  break;
      case 56 : hz = 415;  key = 2;  speed = 3;  break;
      case 57 : hz = 440;  key = 2;  speed = 1;  break;
      case 58 : hz = 466;  key = 2;  speed = 1;  break;
      case 59 : hz = 494;  key = 1;  speed = 2;  break;
```

```
        case 60 : hz = 523;  key = 2;  speed = 3;  break;  // O4C
        case 61 : hz = 554;  key = 2;  speed = 2;  break;
        case 62 : hz = 587;  key = 1;  speed = 1;  break;
        case 63 : hz = 622;  key = 2;  speed = 3;  break;
        case 64 : hz = 659;  key = 2;  speed = 2;  break;
        case 65 : hz = 698;  key = 1;  speed = 3;  break;
        case 66 : hz = 740;  key = 2;  speed = 1;  break;
        case 67 : hz = 784;  key = 2;  speed = 1;  break;
        case 68 : hz = 831;  key = 2;  speed = 2;  break;
        case 69 : hz = 880;  key = 1;  speed = 2;  break;
        case 70 : hz = 932;  key = 2;  speed = 3;  break;
        case 71 : hz = 988;  key = 2;  speed = 3;  break;

        case 72 : hz = 1047; key = 0; break;  // O5C
        case 73 : hz = 1109; break;
        case 74 : hz = 1175; break;
        case 75 : hz = 1245; break;
        case 76 : hz = 1319; break;
        case 77 : hz = 1397; break;
        case 78 : hz = 1480; break;
        case 79 : hz = 1568; break;
        case 80 : hz = 1661; break;
        case 81 : hz = 1760; break;
        case 82 : hz = 1865; break;
        case 83 : hz = 1976; break;

        case 84 : hz = 2093; break; // O6C
        case 85 : hz = 2217; break;
        case 86 : hz = 2349; break;
        case 87 : hz = 2489; break;
        case 88 : hz = 2637; break;
        case 89 : hz = 2794; break;
        case 90 : hz = 2960; break;
        case 91 : hz = 3136; break;
        case 92 : hz = 3322; break;
        case 93 : hz = 3520; break;
        case 94 : hz = 3729; break;
        case 95 : hz = 3951; break;

        case 96 : hz = 4186; break; // O7C

        default : hz = 440;  break;
    }

    serial.send( speed, key );

    return true;
}
```

```
// 鍵盤離した
const bool Sound::midi_stop( unsigned char note, const int sound_num)
{
   if( note_on[ sound_num ] != note ) return false;
   note_on[ sound_num ] = 0;

   return true;
}




////////////////////////////////////////////////////////////////////////////
// SERIAL クライアント


SERIAL::SERIAL(  )
{}


SERIAL::~SERIAL()
{                                                    //fp をクローズ
   tcsetattr(fd, TCSANOW, &oldtio);                  //古いシリアル設定を復帰
   close(fd);                                        //デバイスをクローズ

}


// ソケット作成                              //ソケット作成の代わりに open
void SERIAL::init()
{
   const char dev[] = "/dev/rfcomm0";            //シリアルデバイス


   fd = open( dev, O_RDWR );                     //シリアルデバイスオープン
   if( fd < 0 ){
      perror( dev );
      exit( -1 );
   }

   tcgetattr( fd, &oldtio );                                 //古いシリアル設定を退避

   newtio = oldtio;                                          //新しい設定に古い設定をコピー
   newtio.c_cflag = B9600 | CS8 | CLOCAL | CREAD; //9600 ボーにして 8n1 に設定

   tcsetattr(fd, TCSANOW, &newtio);                  //新しい設定をセット
}
```

```
// データ送信
ssize_t SERIAL::send( int speed, int key )

{
  switch (speed) {
                case 1: write(fd, "1", 1);
                        break;

                case 2: write(fd, "2", 1);
                        break;

                case 3: write(fd, "3", 1);
                        break;

                default: break;
   }


   switch (key) {
                case 0: return write(fd, "a", 1);

                case 1: return write(fd, "c", 1);

                case 2: return write(fd, "b", 1);

   }

}
```

sound.h

```
#ifndef _SOUND_H
#define _SOUND_H

#define MAX_SOUNDS 11 // 最大和音数

#include <pthread.h>
#include <string>
#include "serial.h"                         //serial のヘッダ追加

class Sound
{
   pthread_t thread;
   bool stop_thread;
   int key;
   int speed;

   std::string midi_device;

   // 鍵盤で押しているノート番号
   // 押していない時は 0
   unsigned char note_on[MAX_SOUNDS];

   SERIAL serial;                                   //UDP の udp 作成

public:

   Sound();
   virtual ~Sound();

   void play( const char* device );
   void stop();

   const bool is_playing(){ return (thread ); }

 private:

   static void* launcher( void* args );
   void play_thread();

   const bool midi_play( unsigned char note, const int sound_num );
   const bool midi_stop( unsigned char note, const int sound_num );
};

#endif
```

serial.h

```
#ifndef _SERIAL_H
#define _SERIAL_H

#include <netdb.h>
#include <arpa/inet.h>
#include <string>


//シリアル通信用のヘッダ
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>




class SERIAL
{
   std::string host_to;
   int port_to;

   int soc;
   struct sockaddr_in addr ;

   struct termios oldtio, newtio;       //シリアル通信を行うために追加
   int fd;                              //シリアル通信のファイルポインタ


public:

   SERIAL(  );
   virtual ~SERIAL();

   void init();
   ssize_t send( int speed, int key );
};

#endif
```