第 15 回　みやこ祭

プロジェクションマッピング

引継ぎ資料

資料作成日

2020 年 1 月 4 日

記述内容（関担当分）

配線・ソフトウェア・床面の説明

作成者　18272001　関 優志

（システムデザイン学部　知能機械システムコース）

目次

1. プロジェクションマッピング（A,B,C 面）

1.1 ソースコード

GitHub（下記 URL）よりソースコードをダウンロードできる．ダウンロードできるソースコードは下記の通り．

・Master（コントローラ）のソースコード
・Slave（映像投影用）のソースコード
・PA（音響再生用）のソースコード
・床面（oF と Arduino）のソースコード

GitHub URL

https://github.com/Masashi-Seki/PM_2019_Miyako_Fes

【注意】
全プログラムで addon「ofxOsc」を，Master ではフォントデータ「verdana.ttf」を使用する．

1.2 システム構成

2019 年度のシステム構成を以下に示す．

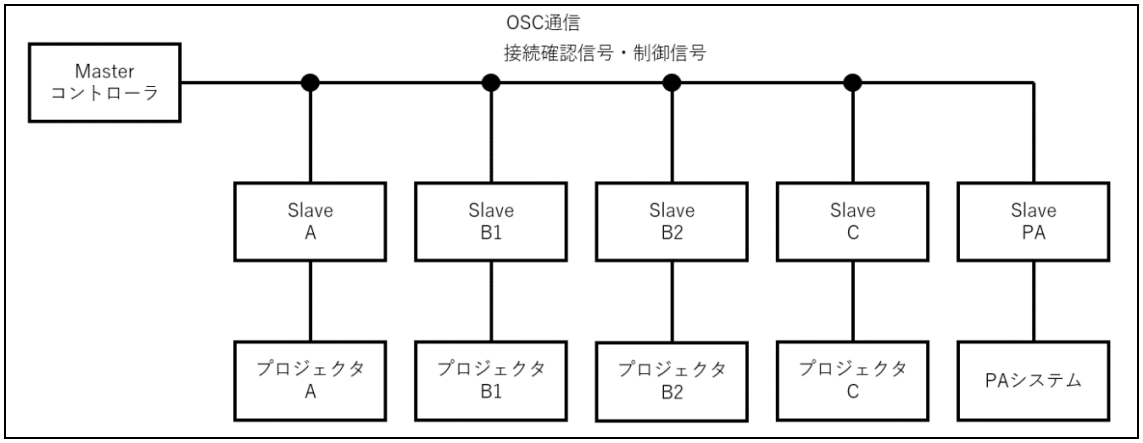

図 1.1　システム構成

映像・音響全体を制御する Master（コントローラ）1 台と，Master からの指令に応じてプロジェクタまたは PA システムに信号を出力する Slave 5 台から構成される．今年度は，コントローラ用 PC に Windows を，PA と映像出力用の PC には Mac を使用した．Master と Slave 間は LAN ケーブルで接続し，Ethernet と OSC 通信を用いて各システムの同期を図っている．Master/Slave のソフトウェア開発には openFrameworks を使用した．

1.3 機器の接続

使用する機器は図 1.2 のように接続する.

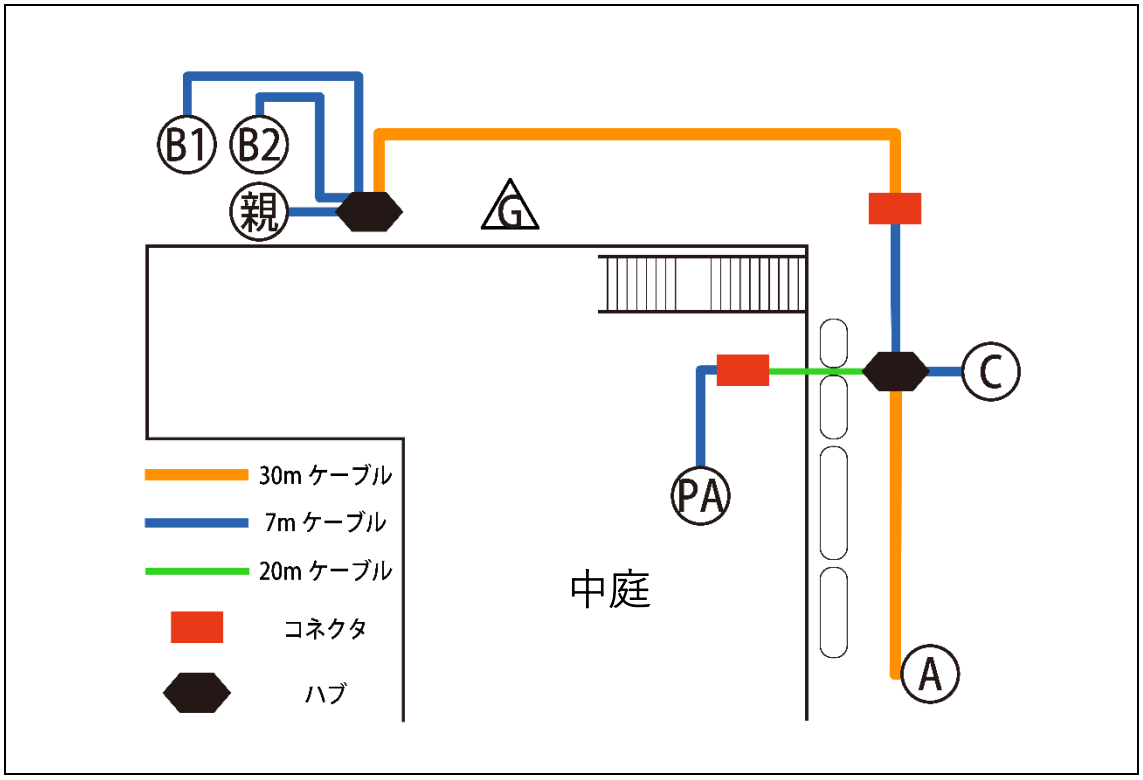

図 1.2　機器の接続

【記号】

・A，B1，B2，C：映像投影用 PC（mac）

・PA：音響用 PC（mac）

・親：コントローラ用 PC（windows）

・G：床面投影機材

1.4 操作手順

1.4.1　接続と確認

接続確認機能により，接続ミスがなく，Master が各 Slave との通信に成功していることを確認できる．これにより，投影開始時になって接続不良が発覚することを防ぐ．

① Master と各 Slave を LAN ケーブルで接続する．
② Master と各 Slave でプログラムを実行する．
③ コントローラ画面の「Connection Check」には「failed」と表示されている．
④ コントローラのキーボードで「c」を押す．
⑤ 「connecting…」と表示され，接続試行中となる．
⑥ 通信が成功すれば，「success」と表示される．

「failed」と表示される場合は，正常に通信が出来ていないことになるので，接続を再確認したり，プログラム上の IP アドレスの設定を確認したりする．

【注意】
・WiFi のポップアップが表示されないように，PC の WiFi 接続は切っておく．

1.4.2　投影

Master（コントローラ）の操作により，全ての Slave を操作することができる．以下に，操作の種類を示す．

表 1.1　コントローラで可能な動作

動作	キー	説明
再生（本編動画）	スペース	本編動画と本編用の曲を再生する．
一時停止	p	本編動画，本編用の曲，QR 用動画，QR 用の曲を一時停止する．
巻き戻し	r	本編動画，本編用の曲，QR 用動画，QR 用の曲を巻き戻す．
映像の表示・非表示	q	全プロジェクタ映像の表示・非表示を切り替える．
再生（QR コード）	g	QR 用動画と QR 用の曲を再生する．

再生中に「q」を押した場合，画面が暗くなり映像は投影されなくなるが，画面への出力を遮断しているだけであり，裏では再生が続いているため，注意する．また，各 Slave でもキー操作により，個別に映像の表示・非表示を切り変えることができる．

表 1.2　各 Slave で可能な操作

動作	キー	説明
映像の表示・非表示	q	全プロジェクタ映像の表示・非表示を切り替える.

なお，操作の優先度が高いのは，Master（コントローラ）である.
（例）Slave 側で映像を非表示にしても，コントローラ側の操作次第で映像を表示可能にできる.

【注意】
・映像が最後まで再生されても自動的に巻き戻しはされない. 続けて再生する場合は「r」を押して巻き戻す.

1.4.3　詳細説明

・OSC メッセージの構成

PC 間は Ethernet という規格（よく見る LAN ケーブルを使ったコネクション）で接続され，OSC 通信という通信プロトコルを使用して通信を行っている. OSC 通信では，インターネットで目にする URL のように送受信する情報をラベリングできる. 今回のシステムで使用した OSC メッセージを図 1.3 に示す.

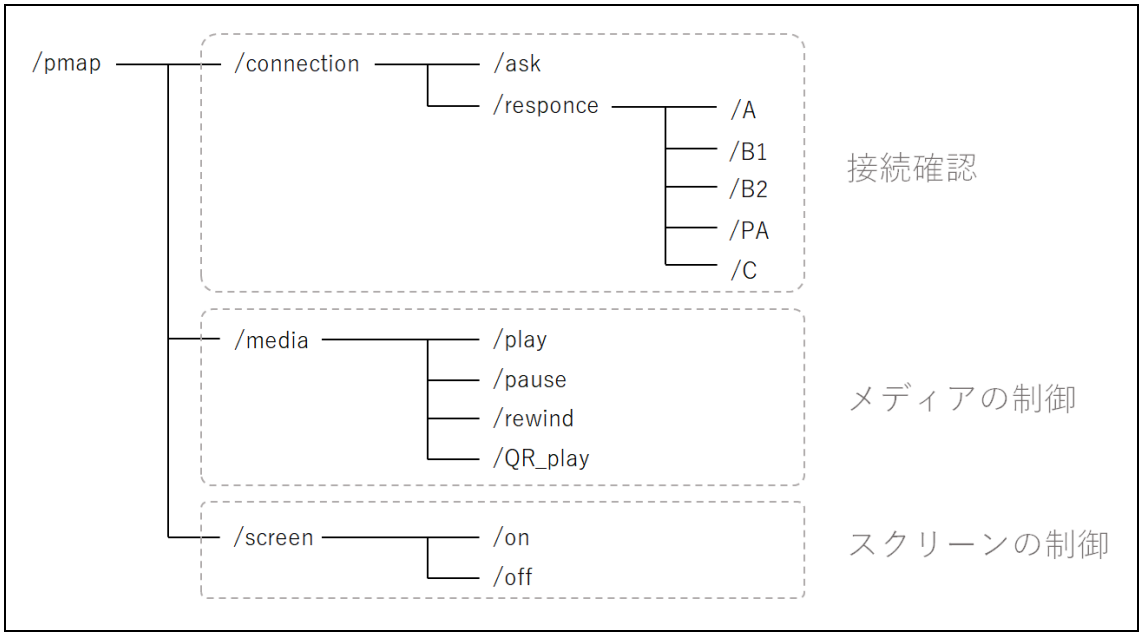

図 1.3　OSC メッセージ

今回定義した OSC メッセージの階層構造は大きく 3 つに分かれている. 通信時は基本的に Master から Slave 方向に送信されるが，接続確認の「/response/**」のみ Slave から Master 方向に送信される.（**：/A，/B1，/B2，/C，/PA のいずれか）

・接続確認のしくみ

1.4.1 に示したように，このシステムでは，Master が各 Slave との通信に成功していることを予め確認することができる．そのアルゴリズム（PC 同士がどのようなやり取りを行って通信成功か否かを判断しているか）を簡単に図 1.4 に示す．

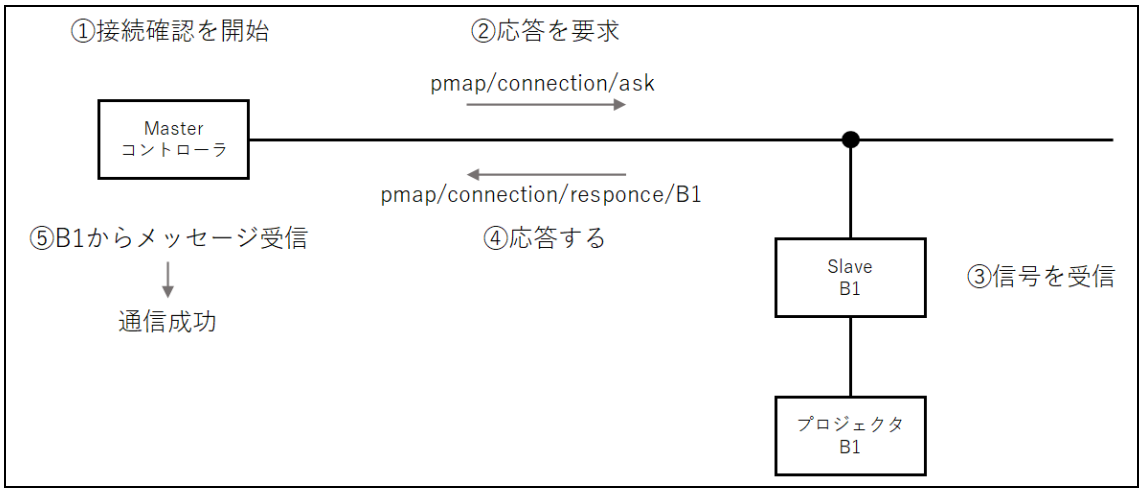

図 1.4　接続確認のしくみ

例として，Master と Slave B1 間の接続チェックの場合を説明する．Master で接続確認処理を開始すると(①), 通信バス(LAN ケーブル)に OSC メッセージ「pmap/connection/ask」が流される（②）．各 Slave は通信バスに対して並列に接続されているため，Master からのメッセージはすべての Slave が受信することができる．Slave B1 は，このメッセージを受信すると，末尾に Slave 名の付いた OSC メッセージを Master に向けて送り返すようにプログラムされている．Slave B1 であれば「pmap/connection/response/B1」を返信する．「pmap/connection/response/B1」を Maste が受信すれば，Master と Slave B1 間でメッセージの受け渡しが成功したことになり，通信路の確立が確認されるから，接続成功と判断している．Master が「pmap/connection/ask」を送信し一定時間が経過しても Slave から応答がないと判断した場合は，処理をタイムアウトさせ「Connection Check」欄に「failed」と表示させる．

1.5 ソースコードの説明

操作する必要のある設定部分のみ説明する．図の左に表示されているのは，ソースファイル内の行番号である．

1.5.1 Master

```
21  #define IP_PROJECTOR_A1 "192.168.11.2"
22  #define IP_PROJECTOR_B1 "192.168.11.3"
23  #define IP_PROJECTOR_B2 "192.168.11.4"
24  #define IP_PROJECTOR_C1 "192.168.11.5"
25  #define IP_PA     "192.168.11.6"
```

ソースコード 1.1　ofApp.h（master）

21～25 行目：各 slave の IP アドレスを設定する．

1.5.2 Slave

```
24  #define IP_CONTROLLER "192.168.11.1" //IP address of controller
25
26  #define PLAY_ADJUST 0 //frames
```

ソースコード 1.2　ofApp.h（slave）

24 行目：Master の IP アドレスを設定する．
26 行目：指令を受け取ってから何フレーム後に処理を実行するか設定．（Slave ごとに再生のタイミングなどの微調整が可能．詳しくは 1.6.4 の「同時再生時に映像のタイミングがずれる」を参照．）

```
25    ofSetFrameRate(60);
26
27    movie.load("movie1.mov");
28    qr.load("movie2.mp4");
```

ソースコード 1.3　ofApp.cpp（slave）

25 行目：映像のフレームレートを設定．書き出した映像に合わせる．
27 行目：本編動画をロード．ファイル名を記入する．
28 行目：QR コード用の動画をロード．ファイル名を記入する．
【注意】
動画・曲データは，oF のプロジェクトフォルダ内の bin > data フォルダの中に入れる．

1.5.3 PA

```
24  #define IP_CONTROLLER "192.168.11.1" //IP address of controller
25
26  #define PLAY_ADJUST 10 //frames
```

ソースコード 1.4　ofApp.h（PA）

24 行目：Master の IP アドレスを設定する．
26 行目：指令を受け取ってから何フレーム後に処理を実行するか設定．（slave ごとに再生のタイミングなどの微調整が可能．詳しくは 1.6.4 の「同時再生時に映像のタイミングがずれる」を参照．）

```
27      sound.load("sound1.mp3");
28      qr.load("sound2.mp3");
```

ソースコード 1.5　ofApp.cpp（PA）

27 行目：本編の曲をロード．ファイル名を記入する．
28 行目：QR コード用の曲をロード．ファイル名を記入する．

【注意】
動画・曲データは，oF のプロジェクトフォルダ内の bin > data フォルダの中に入れる．

1.6 詳細説明（slack で共有したことのまとめ）

1.6.1 IP アドレスの設定

りんごマーク＞システム環境設定＞ネットワーク＞「Ethernet」を選択＞「IPv4 の設定」を「手入力」に変更する．PC ごとに例えば下記のように設定する．
192.168.11.1（コントローラ）
192.168.11.2（A）
192.168.11.3（B1）
192.168.11.4（B2）
192.168.11.5（C）
192.168.11.6（PA）

【注意】
上 2 桁は「192.168.〜」にしないと通信できないことがあったので注意．

1.6.2 うまく通信できない

・mac によって LAN と USB の変換コネクタに相性がある．白と黒があるので，差し替えたりしてみる．
・ハブの電源入っているか確認（ハブにも AC アダプタを挿す必要あり）．
・白の変換コネクタは PC にドライバのインストールが必要(黒は不要のはずだが、細長いやつは使えない)．しかし，macOS が Sierra までしか対応していないので，実質インストールが不可能．ドライバ不要な変換コネクタを調査して 3〜5 つほど購入．

1.6.3 子機でも映像再生できるようにしたい

子機（slave）のプログラム ofApp.cpp の keyPressed 関数内のコメントアウトを外す．
'p'：再生
's'：停止
'r'：巻き戻し
本番はコントローラから制御するので，使わない．

1.6.4 フルスクリーン表示すると映像がずれる

映像の書き出しの時のアスペクト比があっているか確認．

1.6.5 フルスクリーンにならない

子機（Slave）のプログラム main.cpp の中，①をコメントアウトして，②のコメントアウトを外す．

```
16  #include "ofMain.h"
17  #include "ofApp.h"
18
19  //========================================================================
20  int main( ){
21    ofSetupOpenGL(1024,768,OF_WINDOW);  ①
22    //ofSetupOpenGL(1024, 768, OF_FULLSCREEN);  ②
```

ソースコード 1.6　main.cpp（slave）

1.6.6 親機だけ通信がうまくいかない

今年は親機だけ Windows を使用した．この時は，oF のアドオン ofxOsc が VisualStudio でうまく読み込めていなかったことが原因で，プロジェクトを立ち上げ直したら解決した．

1.6.7 同時再生時に映像のタイミングがずれる

子機（Slave と PA）のプログラム ofApp.h の中，①の数値を変更する．（デフォルトは 0）ここで設定したフレーム数だけ，制御信号が届いてから映像を再生または停止させる処理を【遅らせる】ことができる．例えば，30fps で出力した映像を再生する場合，1/30sec 単位で調整できることになる．

```
26   #define PLAY_ADJUST 0 //frames ———————— ①
```

ソースコード 1.7　ofApp.h（slave と PA）

1.6.8 プログラムを実行すると勝手に再生が始まる

【症状】

・プログラムを実行すると勝手に本編動画の再生が始まる．

・初めの 1 回だけ Slave のキーボード操作により手動で停止させれば，以降は勝手に再生されることはなく，Master からの通信により制御可能．

【原因】

本番直前まで原因が不明であった．10/31 に使用する PC に対して調査してもらい，原因は oF のバージョンで nightly 版を使用していることである可能性が高いことが分かった．そもそも nightly 版を使ったのは，Catalina で Xcode と(nightly 版ではない)oF をダウンロードして試したところ，プロジェクトを作成して実行する（プログラムは書いていない）段階でエラーが出てしまう症状があり，nightly 版を使うことでこの症状が解決することが分かったためである．原因が発覚したのが本番直前だったため，今年はこのまま nightly 版を使う方針でいくことになった．

表 1.3　PC の挙動調査（2019/10/31）

名前	PC	担当	親機を受け付けない	音が勝手に流れる	親機から操作ができるが送信しない	Xcode	OF	OS
宮崎ひとみ	Windows	親機						
	Mac	PA	異常なし	異常なし	異常なし	10.1	10.1	Mojave
北野	Mac		異常なし	異常あり	異常なし	10.1	nightly	HighSierra
モルト	Mac		異常なし	異常あり	異常なし	11.1	nightly	Catalina
宮崎優子	Mac		異常なし	異常なし	異常なし	10.1	0.9.8	HighSierra
鈴木	Mac		異常なし	異常あり	異常なし	11.1	nightly	Mojave
武藤	Mac		異常なし	異常あり	異常なし	10.1	nightly	Mojave

【対処法】

とりあえず，今年度乗り切る（った）対策として，Master からの指令が無いのに動画が再生されている場合は，再生を停止させるコードを追加した．これにより，nightly 版でも勝手に再生が始まる症状は見かけ上防げた．

1.7 今年の問題点

今年はシステムまわりに手を付ける時期が遅く，発生した問題に対処する時間が十分に確保できなかった．(2018 年度通りならスムーズにできると思ったが，バージョン関係の問題で結構詰まった．）結局本番でも通信が安定せず，1 台だけ映像が再生されないなどの問題が発生した．

1.8 来年やるべきこと

来年は早々 (夏休み中！) にシステムまわりが安定動作するところまで確認するべき．(問題があっても時間があるので解決 or 対処できる可能性が高いので．)

macOS については，Mojave 以下が望ましいが，OS を引き下げてまでやる必要はないのではないかとの見解あり．

2　プロジェクションマッピング（床面）

USB ケーブルでパソコンに接続できるコントローラと，プロジェクタで投影する映像のプログラムを製作した．運営側が人の動きに合わせてコントローラを操作し，あたかも人の動きにより映像が変化しているような体験を提供できる． PC 側のプログラム（oF や Processing）をシリアル通信に対応させれば，コントローラの操作により映像に変化を与えることができる．

2.1. ハードウェア

2.1.1. システム構成

コントローラは，USB ケーブルでパソコンに接続して使用する．コントローラに内蔵されたマイコンが，押されたボタンの情報をシリアル通信でパソコンに送信する．また，パソコンから信号を送信することで，コントローラの LED を制御し，ボタンの点灯・消灯などを制御することもできる．これらの入出力信号と連動した映像データをプロジェクタに出力している．

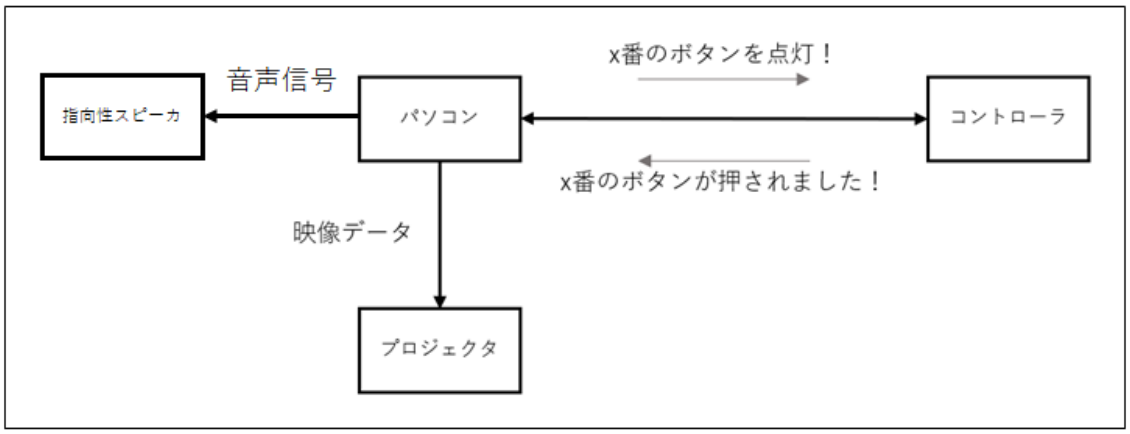

図 2.1　システム構成

ボタンの番号は図 2.2 のように定義している．また，図 2.3 のように，16 個のボタンそれぞれの下に，タクトスイッチ 2 つとフルカラーLED が設置されている．（コントローラの天板は接着していないため外すことができるが，（3D プリントした）ボタンがバラバラに取れるため注意！）

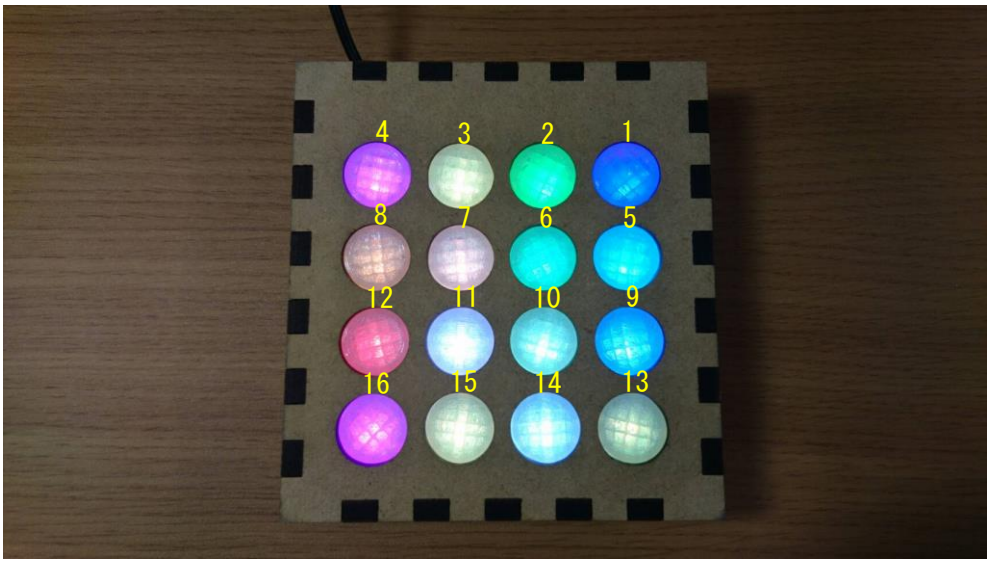

図 2.2　ボタン番号の定義

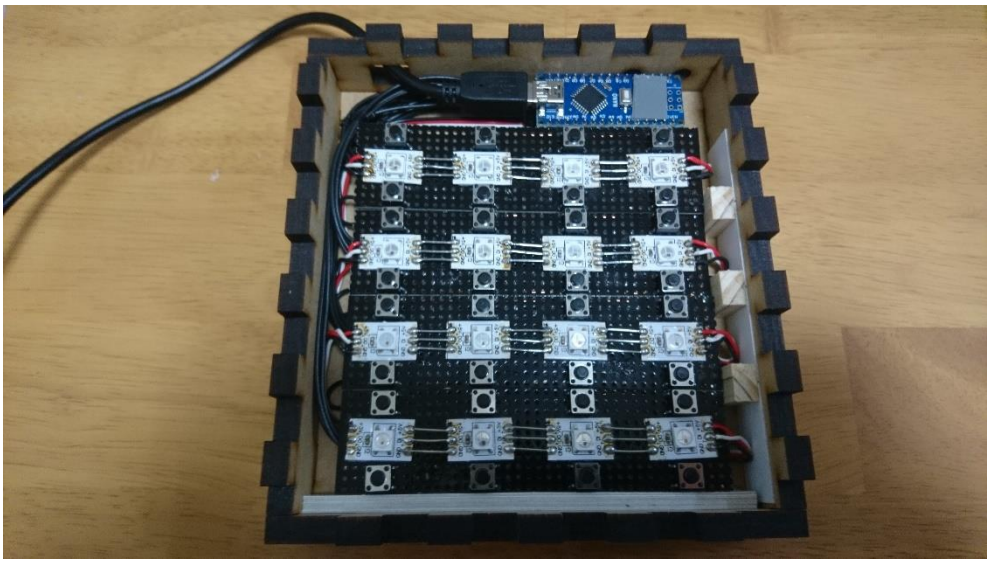

図 2.3　コントローラ内部

2.1.2. 回路図

図 2.4 に，コントローラの回路図を示す．各ボタンの下にはスイッチが 2 つずつ設置されており，どちらか，もしくは両方のスイッチが押されると信号線が GND に接地し，マイコンが認識する．フルカラーLED には個別で色を制御可能な NeoPixel を使用した．

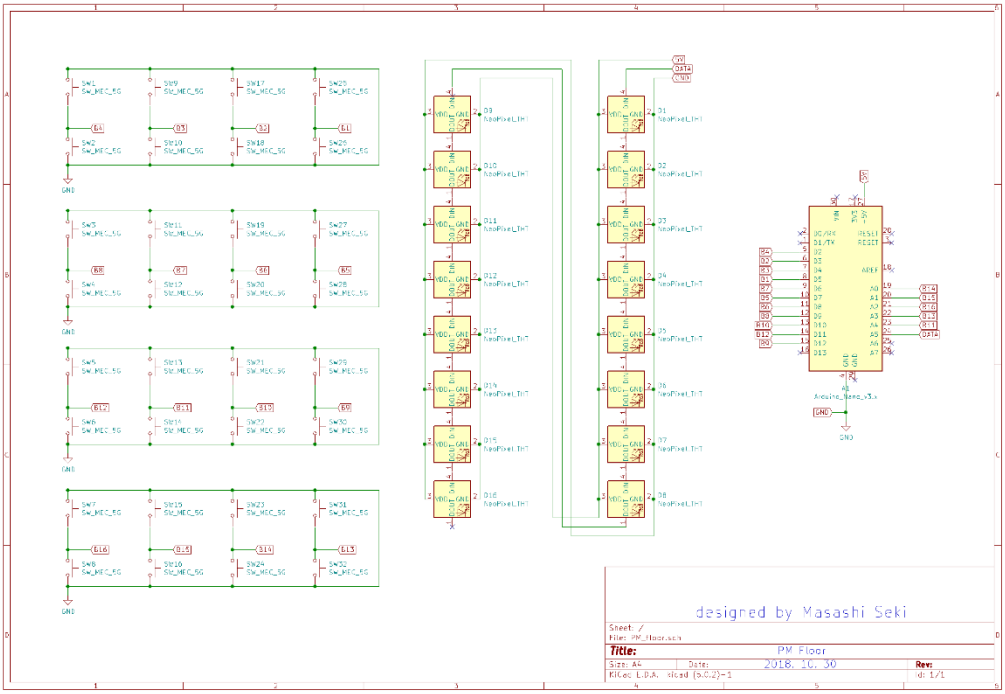

図 2.4　コントローラ回路図

2.1.3. 指向性スピーカの導入

今年度の新たな試みとして，指向性スピーカを使ってみた．比較的安価に手に入る秋月電子の組み立てキットを使用した．

・秋月電子　パラメトリック・スピーカー実験キット
http://akizukidenshi.com/catalog/g/gK-02617/

・秋月電子　パラメトリック・スピーカー増設キット
http://akizukidenshi.com/catalog/g/gK-02815/

指向性スピーカは通常のスピーカと異なり，スピーカを向けた方向にピンポイントで音を届けることが可能である．今回は，プロジェクタの横（2 階）から地面に向けて設置することで，作品の中に入り込むと効果音が聞こえるという不思議な体験を提供することができたと思う．また，指向性スピーカは遠距離からでも十分な音量を届けられるため，PC の近くに設置でき，長い配線を使用しなくてよいという利点がある．

※私物として買ったので，使いたい方は関まで教えてください．

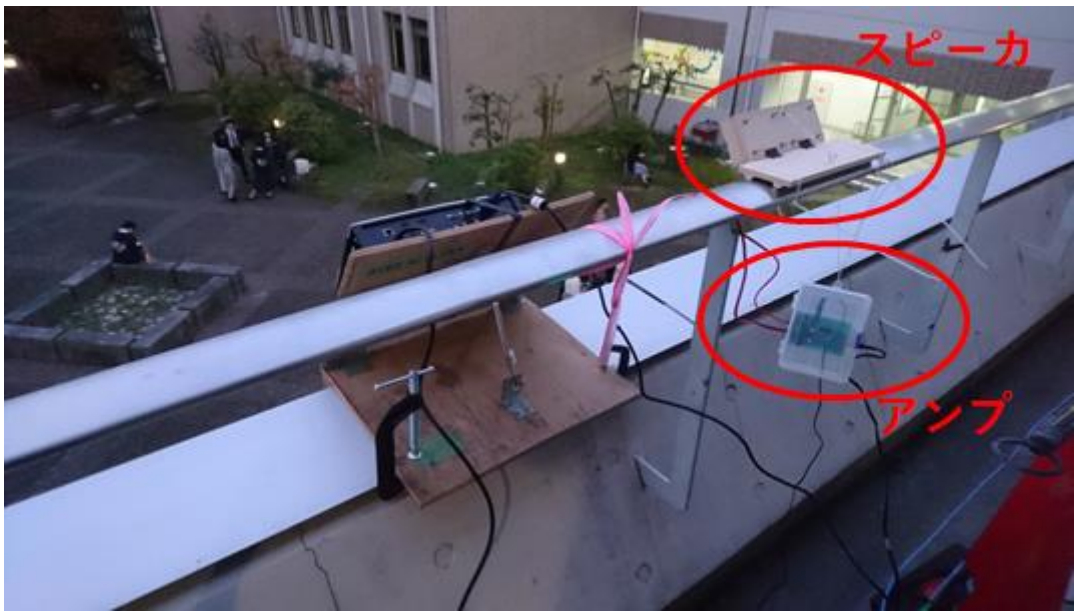

図 2.5　スピーカの設置位置

2.2. ソフトウェア解説

パソコンとコントローラ間はシリアル通信により，数値データを互いに送受信している．パソコンとコントローラの各プログラム内で，受信した数値により実行する処理を記述している．

2.2.1. コントローラ側

コントローラのプログラミングには Arduino を使用している．使用しているマイコンボードは Arduino Nano である．コントローラの行っている処理は次の 2 つである．

① パソコンから数値データを受信し，受信したデータによりボタンの LED を変化させる．

② ボタンの状態をチェックし，押されたボタンの番号をパソコンに送信する．

以下に，コントローラ側のプログラムの処理について説明する．

```
#include "button.h"
```

ボタンと Arduino のポート番号の対応関係を記述した「button.h」を使用するためにインクルードしている．

```
#include <Adafruit_NeoPixel.h>
```

NeoPixel LED を使用するために必要なライブラリのインクルード. Adafruit_NeoPixel.h をダウンロードしてパソコン入れないと使用出来ないので注意. LED 関係の処理については，ライブラリをダウンロードすると入っているスケッチの例「simple」を参照すると分かりやすいと思う．

```
#define PIN    19
#define NUMPIXELS    16
```

Arduino に接続されている LED の DATA 線のピン番号と LED の数の設定．ここは変更なしで OK.

```
double rate = 0.2
```

LED の明るさの指定. LED は 0～255 の範囲で明るさを指定できるが，かなり明るいので，20%くらいの出力で対応した．

```
int color[16][3]={
        {0 * rate, 0 * rate, 255 * rate},
        {0 * rate, 255 * rate, 0 * rate},
        …
}
```

16 個のボタンの色を設定している．3 つの数値はそれぞれ RGB である．明るさ調整のために上述の変数「rate」をかけている．この色の設定が OpenFrameworks 側の円の色と対応しており，映像データとボタンで配置を統一している．

```
Serial.begin(9600)
```

シリアル通信の開始．現在は通信速度 9600bps を使用しており，これはパソコン側と同じ通信速度に指定する必要がある．

```
push1 = buttonCheck()
push2 = buttonCheck()
```

ボタン状態の取得．「buttonCheck()」は，押されているボタンの番号を返す関数である．押されていない時は 0x00 を返す．

```
if (push1 == push2) {…}
```

ボタン処理に入る．2 回取得したボタン情報が同じであれば，ボタンを押された時の処理に入る．わざわざボタン状態を 2 回取得して比較しているのは，ノイズやチャタリングによる誤動作を防ぐためである．

```
Serial.write(0x01)
```

パソコンに数値データを送信する．「Serial.write()」の引数に指定した数値をシリアル通信でパソコンに送信する．上記の例なら，数値 0x01 を送信している．

```
if (push2 == 0x01)
else if (push2 == 0x02)
else if (push2 == 0x03)
…
```

Arduino 上で押されたボタンを識別する．変数「push1」と「push2」には押されたボタンの番号が入っているので，push2 の値により条件分岐している．

```
if (Serial.available() > 0) {…}
```

パソコンから Arduino に送られてきた数値データを確認する．「Serial.available()」は，Arduino がデータを受信すると 1 を返すので，この if 文は Arduino がデータを受信した時に実行される．

```
getData = Serial.read()
```

コントローラが受信したデータを取得する．データを受信したときに「Serial.read()」を実行すると，戻り値として1byteの受信データを得ることができる．

```
if (getData == 0x01)
else if (getData == 0x02)
else if (getData == 0x03)
```

```
…
```

コントローラが受信した数値データにより，コントローラが行う処理を決めている．パソコンから送られてきた数値は変数「getData」に入るため，この変数の値で条件分岐している．

また，本プログラムには以下の関数が用意されており，組み合わせることでプログラミング可能である．

```
reset_LED()
```

全ての LED を点灯する．

```
allOff_LED()
```

全ての LED を消灯する．

```
off_LED(int no)
```

引数に指定した番号の LED を（個別に）消灯する．

```
on_LED(int no)
```

引数に指定した番号の LED を（個別に）点灯する．

```
flash_LED(int no)
```

引数に指定した番号の LED を（個別に）点滅させる．

```
buttonCheck()
```

コントローラのボタン状態を取得する．ボタンが押されているときはそのボタン番号を返し，押されていない時は 0x00 を返す．

```
LED_blink()
```

全ての LED を 3 回点滅させる．

2.2.2. パソコン側

パソコン側は OpenFrameworks により処理をしている．パソコン側の行っている処理は次の 3 つである．

① コントローラから数値データを受信し，受信したデータにより映像を変化させている．
② コントローラの LED 状態を変化させるために数値データを送信している．
③ キーボードの状態をチェックし，各キーに割り振った数値データをコントローラに送信する．

以下に，パソコン側のプログラムの処理について説明する．

```
ofHideCursor()
CGDisplayHideCursor(NULL)
```

実行画面にカーソルを表示させないための関数．Windows の場合は「ofHideCursor()」を，Mac の場合は「ofHideCursor()」だけでは消えないことがあるため，「CGDisplayHideCursor(NULL)」も記述する．

```
serial.setup("COM3", 9600)
serial.setup("/dev/tty.usbmodem1411",9600)
```

シリアル通信のポートと通信速度の設定．関数の引数は，ポート番号と通信速度である．コントローラを接続してArduinoを認識したCOMポートの番号（場所）に合わせる．通信速度はパソコンとコントローラで合わせる必要があり，現在は9600bpsとなっている．Windows の場合は「serial.setup("COM3", 9600)」のように，Mac の場合は「serial.setup("/dev/tty.usbmodem1411",9600)」のように記述する．

```
getData = serial.readByte()
```

パソコンが受信したデータを取得する．データを受信したときに「serial.readByte()」を実行すると，戻り値として1byteの受信データを得ることができる．

```
serial.writeByte(0x01)
```

コントローラに数値データを送る．「serial.writeByte()」の引数に指定した数値をシリアル通信でコントローラに送信する．上記の例なら，数値0x01を送信している．

```
if (getData == 0x01) {
        cout << "button1 pushed" << endl;
        serial.writeByte(0x01);
}
…
```

OpenFrameworksで，押されたボタンを識別する．getDataに入った数値が押されたボタン番号に対応しているので，if文で条件分岐すればよい．ここでは，ボタンが押されるとコンソール画面に「button x pushed」と表示し，コントローラにボタンの番号を送り返している．（つまり，コントローラのボタンを押すと，そのボタンのLEDが消灯する．）

以上