令和４年度卒業研究報告書

心拍センサーを用いたアート作品の改良と機能拡張

情報技術科　古屋　勇翔

昆野　捷冴

指導教員　　菅野　研一

目次

第 1 章　はじめに 3

第 2 章　研究概要 4

2.1 作品概要 4

2.2 システム構成 5

第 3 章　開発環境 6

3.1 Arduino-IDE 6

3.2 Pure Data 6

第 4 章　通信 7

4.1 Wi-Fi 7

4.2 UDP 7

4.3 OSC 8

第 5 章　ハードウェア 9

5.1 ESPr Developer 32 9

5.2 LED 11

5.3 心拍センサー(脈波センサー) 12

5.4 ロジックレベルコンバーター 13

5.5 ステレオミニプラグケーブル 14

5.6 コネクタ・基板 15

第 6 章　ソフトウェア 18

6.1 脈波について 18

6.1.1　脈波の取得 18

6.1.2　値の平滑化 19

6.1.3 閾値の設定.....20
6.1.4 波の検出.....21
6.1.5 データの保存.....22
6.1.6 類似度判定.....22
6.2 LED 処理について.....26
6.2.1 Adafurit_NeoPixel ライブラリ.....26
6.2.2 光の種類について.....26
6.2.3 色の処理について.....27
6.3 Pure Data(以下 Pd).....29
6.3.1 ライブラリ.....29
6.3.2 変数.....30
6.3.3 Wi-Fi と UDP 接続.....31
6.3.4 OSC 通信.....32
6.3.5 機能.....32
6.3.6 主なオブジェクト.....33
6.3.7 インレット・アウトレット.....34
6.3.8 パッチの作成方法.....35
6.3.9 受信と音の出力.....38
6.3.10 音の種類について.....42
第 7 章 終わりに.....43
参考文献.....44

第1章　はじめに

過去の卒業研究で「入力デバイスに水を用いて音を生成するアート作品」[1],「入力デバイスに水を用いて音を出力・制御する楽器の製作」[2],「入力デバイスに水を用いて音と光を出力するシステムの製作」[3]がというものあり,「水と向き合う」というアート作品としての出展歴がある.[4][5]以前に産技短展で「水と向き合う」に実際に触れ面白いと感じた.見るだけではなく実際に体験できる展示品の方が,面白味があると思ったため,実際に触れて体験できるアート作品を作成したいと考えた.

昨年の卒業研究で「心拍信号を利用したアート作品の制作」[6]というものがあった.展示をするには改善が必要だと聞いた.そこで今回私たちは,改善をし,展示ができるように作成に取り組むことにした.

第 2 章　研究概要

2.1 作品概要

２台の脈波センサーから読み取った値を計算し,類似度を算出する.算出された類似度によって光と音の演出が変わる.全体図は次のようになっている.

LED はアクリル円筒に貼り付け,PC で音の出力を行っている.

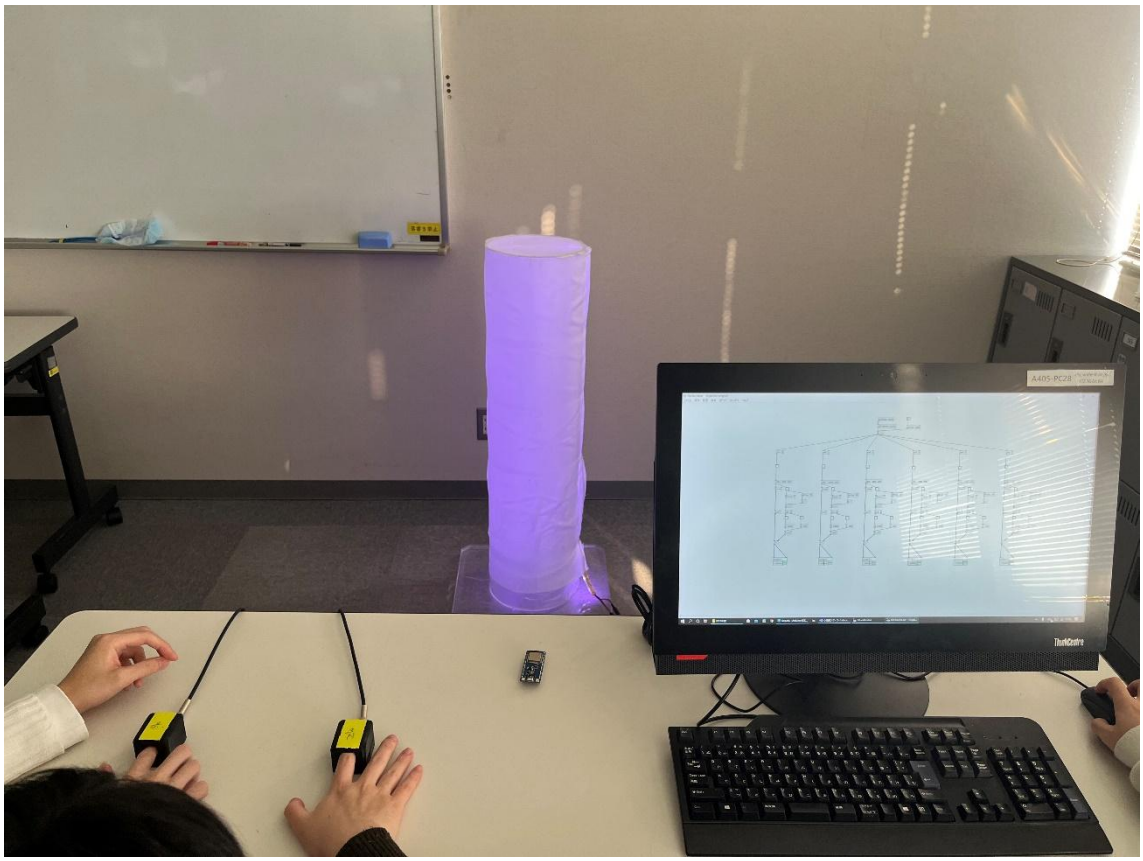

図 1　全体図

2.2　システム構成

主に使用しているものは心拍センサー, ESP32, LED, アクリル円筒, PC,である.

心拍センサーで脈波を取得し,ESP32 で値の処理, LED の制御, PC との無線通信を行っている.

PC は音の出力を行うため使用している.

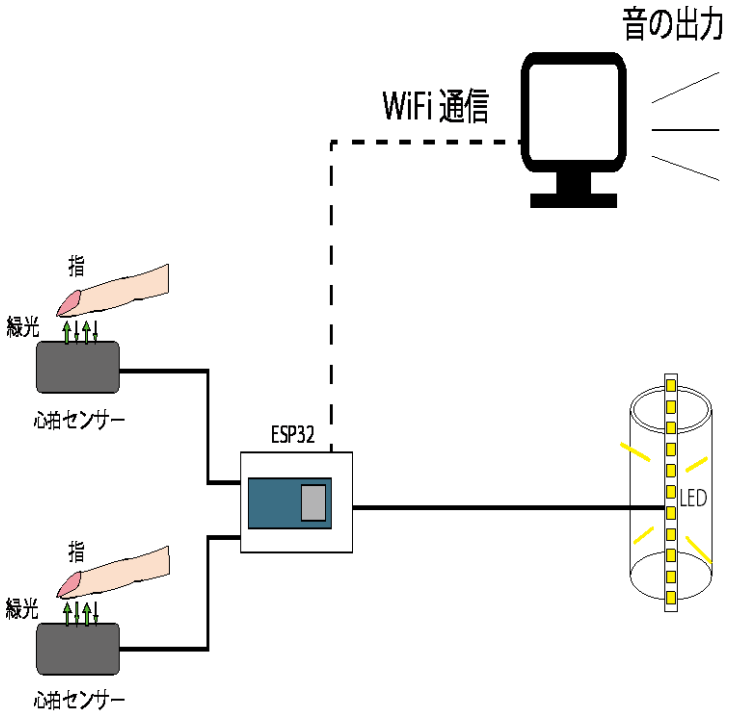

図 2　システム構成図

第 3 章　開発環境

3.1 Arduino-IDE

Arduino-IDE はボード上で動作するソフトウェアを開発するために作られた統合開発環境である.ソースコードの編集や,コンパイル,リンク,作成されたオブジェクトのボードへの書込み機能などが提供されている.

3.2 Pure Data

Pure Data とは,音声,ビデオ,映像処理のためのリアルタイムなグラフィカルプログラミング環境である.フリーソフトであり,どのプラットフォームでも動作し,シンセサイザ,スペクトル解析,三次元映像処理・生成などができる.

Arduino と組み合わせた自作楽器の製作も可能である.

第 4 章　通信

4.1 Wi-Fi

Wi-Fi(Wireless Fidelity)とは,電波を用いた無線通信により近くにある機器間を相互に接続し,構内ネットワーク（LAN）を構築する技術である.

今回は,ESP32 と PC 間で通信をするために Wi-Fi を使用している.

4.2 UDP

UDP(User Datagram Protocol)とは, インターネット・プロトコル(IP)を使ったネットワークにおいて,アプリケーション同士が最小限の仕組みでデータを送受信できるトランスポート層のプロトコルである.TCP に比べてプロトコルが簡素なため,信頼性には劣るが,通信の処理にかかるコストが少ないという性質から,リアルタイム性が重要視される通信で使われている

4.3 OSC

OSC（Open Sound Control）とは,電子楽器（特にシンセサイザー）やコンピュータなどの機器において音楽演奏データをネットワーク経由でリアルタイムに共有するためのアプリケーション層のプロトコルである.OSC は TCP か UDP プロトコルを使用して通信される.一般的には UDP を使用することが多いため,本研究でも UDP を使用する． OSC メッセージ（OSC Message）と OSC 引数（OSC Argument）の 2 つに分けられる．OSC メッセージは，送受信する OSC の情報内容をラベリングしたもので OSC の値が何を意味しているのかを表す．メッセージは階層構造を持つことが可能で，階層を「/」で区切って表現する．OSC 引数は実際の値を送信する．値は，整数，実数，文字列等様々な型を送受信することが可能である．また，複数の値を一度に送受信することもできる． 本研究では,心拍数の類似度をデータとして送受信するために使用している.（図 3）

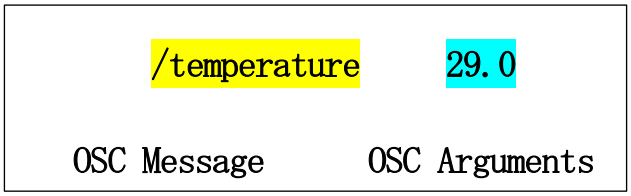

図 3 OSC プロトコルの仕様

第5章　　ハードウェア

5.1　ESPr Developer 32

ESPr Developer 32 は Espressif Systems の無線通信モジュール（Wi-Fi＋Bluetooth）ESP-WROOM-32 と USB-シリアル変換 IC FT231XS を搭載した開発ボードである．その他にも 3.3 V 出力レギュレータやリセットスイッチ，動作モード切替用スイッチが搭載されているので，ESP-WROOM-32 Wi-Fi + Bluetooth モジュールをすぐに使用可能．ESPr Developer 32 の仕様は表，ESP-WROOM-32 の仕様は表の通りである．

表 1　ESPr Developer 32

電源	USB Type-C 5V 電源入力ピン 3.7V~6.0V 電源出力ピン USB または Vin ピンに入力した電圧を出力
動作電圧	3.3V

表 2　ESP-WROOM-32

Wi-Fi	802.11 b/g/n（HT40）
Bluetooth	Classic，BLE 4.2（デュアルモード）
SRAM	520KB
フラッシュ	4MB(32Mbit)
動作電圧	2.2～3.6V

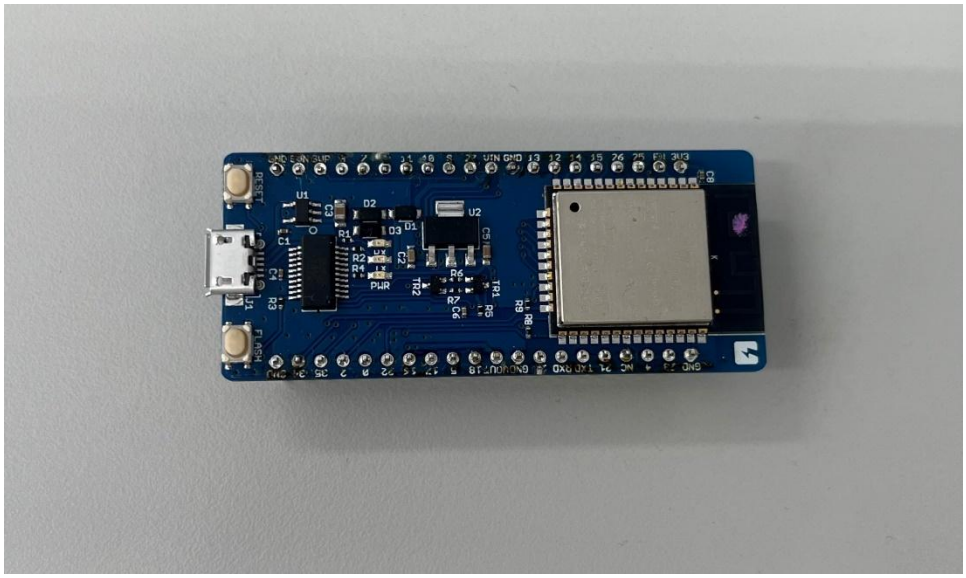

図 4　ESPr Developer 32

5.2 LED

LED は Neo Pixel を使用した.従来からある RGB フルカラーLED は,色は変化はするが,LED テープライト全体が同時に変化する.

Neo Pixel は LED1 個ずつに IC（マイコン）チップが搭載されているため LED の色や明るさを細かくコントロールすることが可能である.今回の研究では画像のように虹色に光らせる演出を取り入れたいため Neo Pixel を使用した.(図 5)

図 5　Neo Pixel

5.3 心拍センサー(脈波センサー)

指先などの末梢血管のある部分を光センサー部に置いて脈波をアナログ出力するセンサーデバイスである.(図 6)脈波とは心臓が血液を送り出すことに伴い発生する,血管の容積変化を波形としてとらえたものである.(図 7)

図 6　　心拍センサー(脈波センサー)

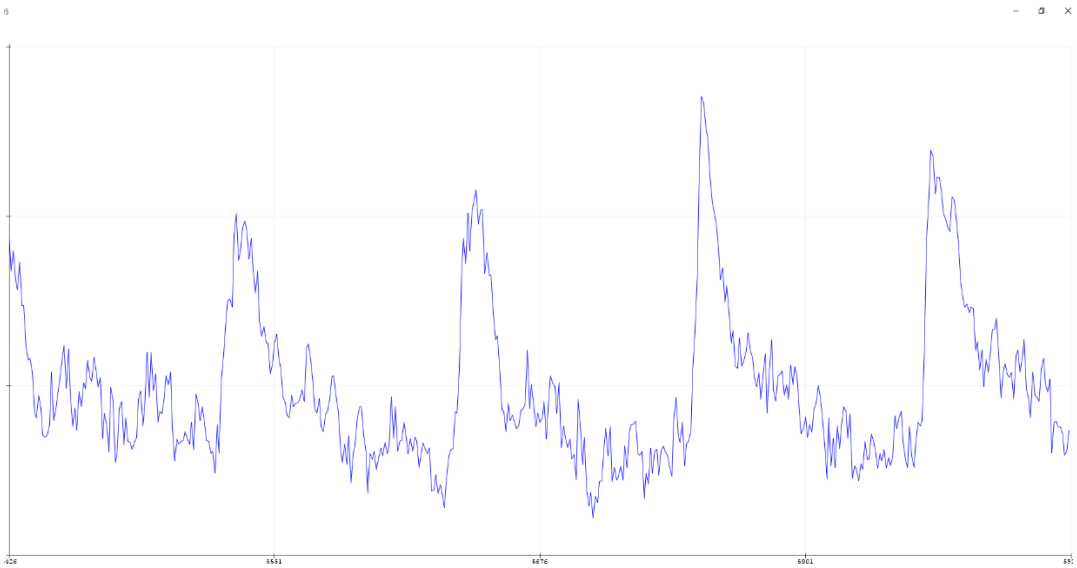

図 7　脈波

5.4 ロジックレベルコンバーター

昨年度の研究の課題として安定して光らせることができないというものがあった.原因の一つとして,ESP32 と Neo Pixel を直接つないでしまったことにあると考えた.その理由として,ESP32 は 3.3V で稼働するのに対して,Neo Pixel は5.0V で稼働するため直接つないでしまうと稼働することもあるが,制御の安定性に欠けていた.そこで,今回の研究では 3.3V から 5.0V へ変換するロジックレベルコンバーターを使用することにした.(図 8)(図 9)

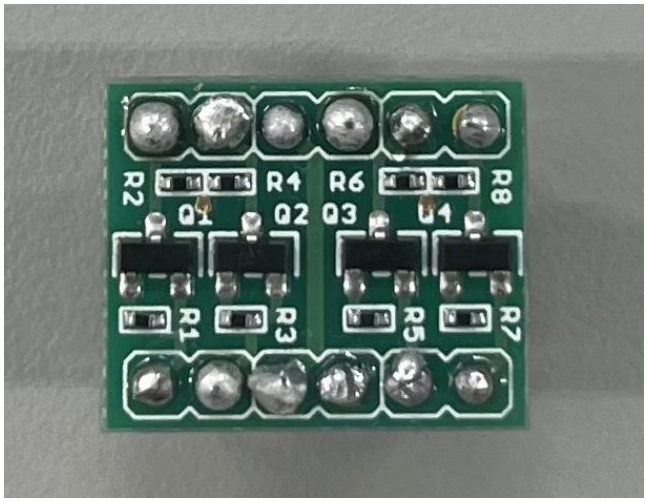

図 8　ロジックレベルコンバーター

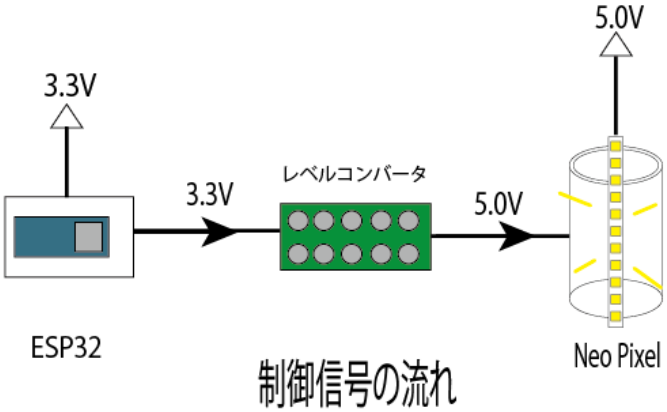

図 9　制御信号の流れ

5.5 ステレオミニプラグケーブル

心拍センサーに電源供給するためにステレオミニプラグケーブルを使用する.(図 10)しかし一部断線していて値を取得しない問題が発生していたため修復した.(図 11)

図 10 ステレオミニプラグケーブル

図 11　修復後のステレオミニプラグケーブル

5.6 コネクタ・基板

展示の準備をしやすくするためにコネクタの改良と基板の作成を行った.(図 12)(図 13)(図 14)

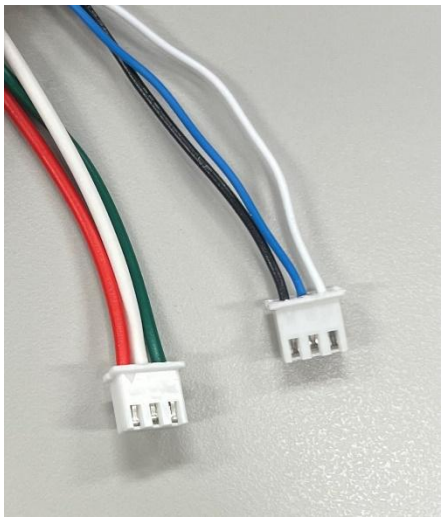

図 12　コネクタ

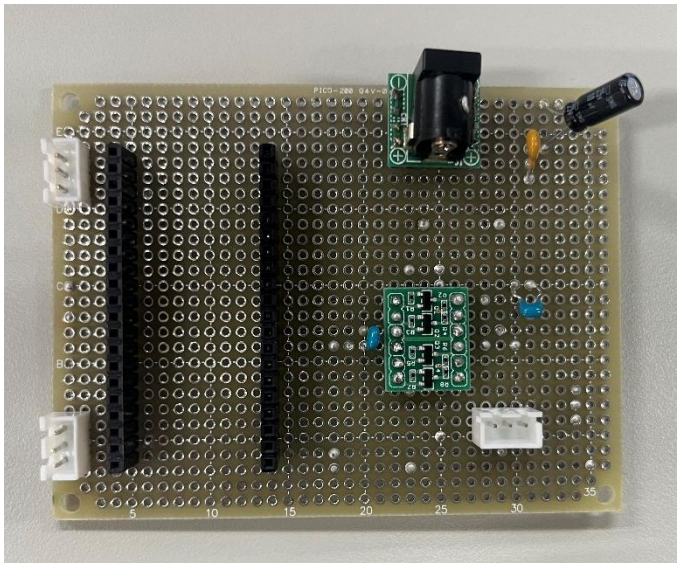

図 13　基板(表)

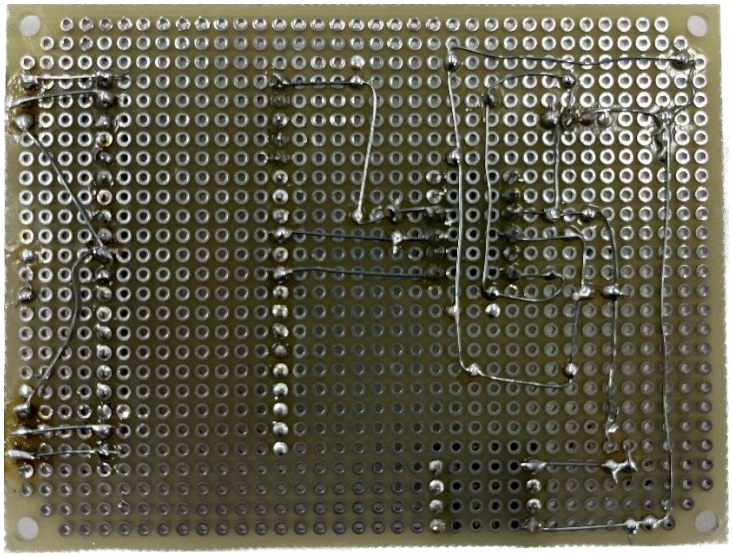

図 14　基板(裏)

今回作成した基板の部品と回路図は下の図のとおりである.(図 15)(図 16)

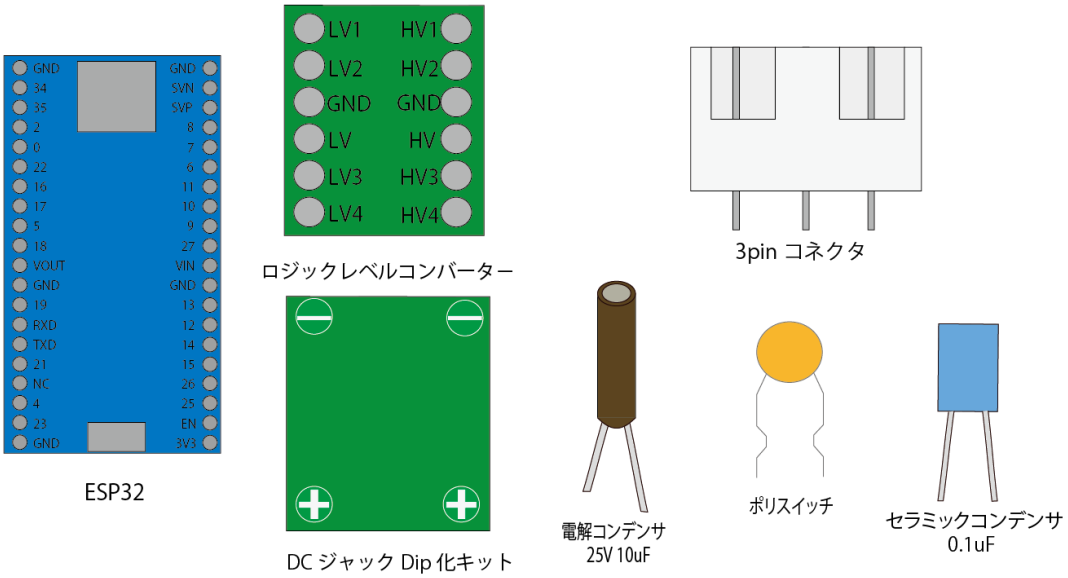

図 15　使用部品

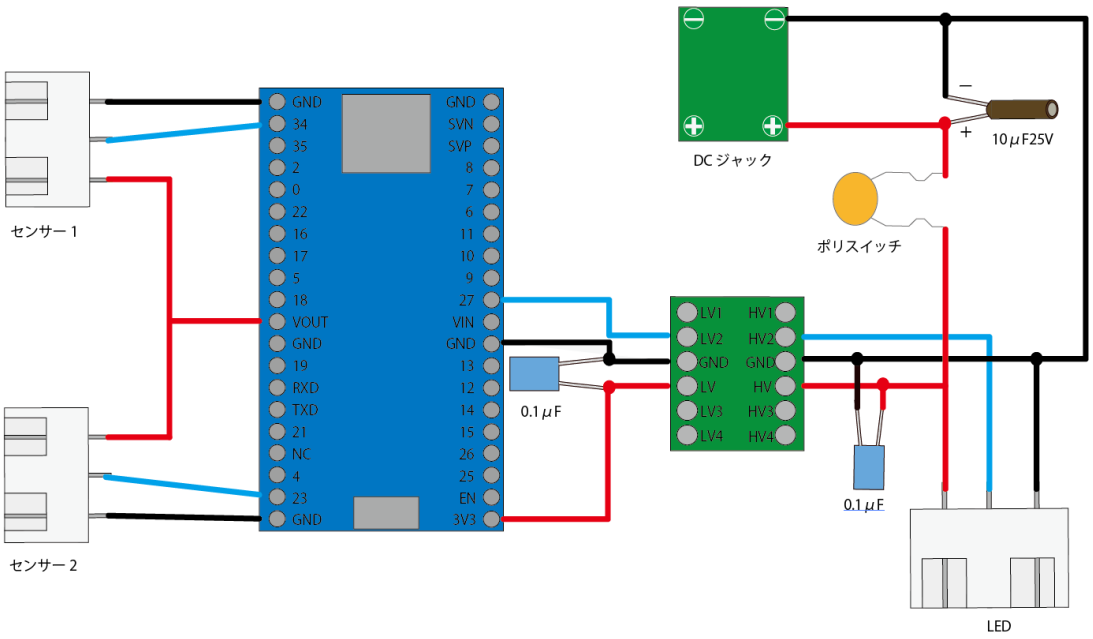
センサー 1
センサー 2
GND
34
35
2
0
22
16
17
5
18
VOUT
GND
19
RXD
TXD
21
NC
4
23
GND
GND
SVN
SVP
8
7
6
11
10
9
27
VIN
GND
13
12
14
15
26
25
EN
3V3
0.1μF
LV1
LV2
GND
LV
LV3
LV4
HV1
HV2
GND
HV
HV3
HV4
DC ジャック
10μF25V
ポリスイッチ
0.1μF
LED

図 16　回路図

第6章　ソフトウェア

6.1 脈波について

心拍を取得するコードが残っていなかったため昨年の研究の報告書を参考に作成した.[6]

6.1.1　脈波の取得

センサーの電源ピン，GND ピン，アナログピンを ESP32 に接続することで，心拍を 0~4095 の電圧値で取得することができる．取得には analogRead 関数を使用する．analogRead 関数は指定したアナログピンからの値を読み取る関数である.

アナログピンを 34 ピンと 35 ピンに設定している．(図 17)ピンから読み取った値はそれぞれ用意した配列に格納していく.データは５０個格納している.(図 18)

```
const int sensorPin1 = 34;
const int sensorPin2 = 35;
```

図 17　ピンの選択

```
for (int i = n; i > 0; i--) f[i % 50] = analogRead(sensorPin1);  //過去50回分の値を記録
for (int i = n; i > 0; i--) g[i % 50] = analogRead(sensorPin2);  //過去50回分の値を記録
```

図 18　値の格納

6.1.2　値の平滑化

脈波センサーから取得した値(RAW データ)のままだとばらつきがあるため値を平滑化する必要がある.平滑化をするのに移動平均という方法を使った.

50 個のデータの総和を求め,平均を算出している.(図 19)

平滑化前とは平滑化後の図 20 のようになっている.(図 20)

```
for (int i = 0; i < n; i++) ave1 += f[i % 50];
ave1 = (int)ave1 / n;
```

図 19　平滑化処理

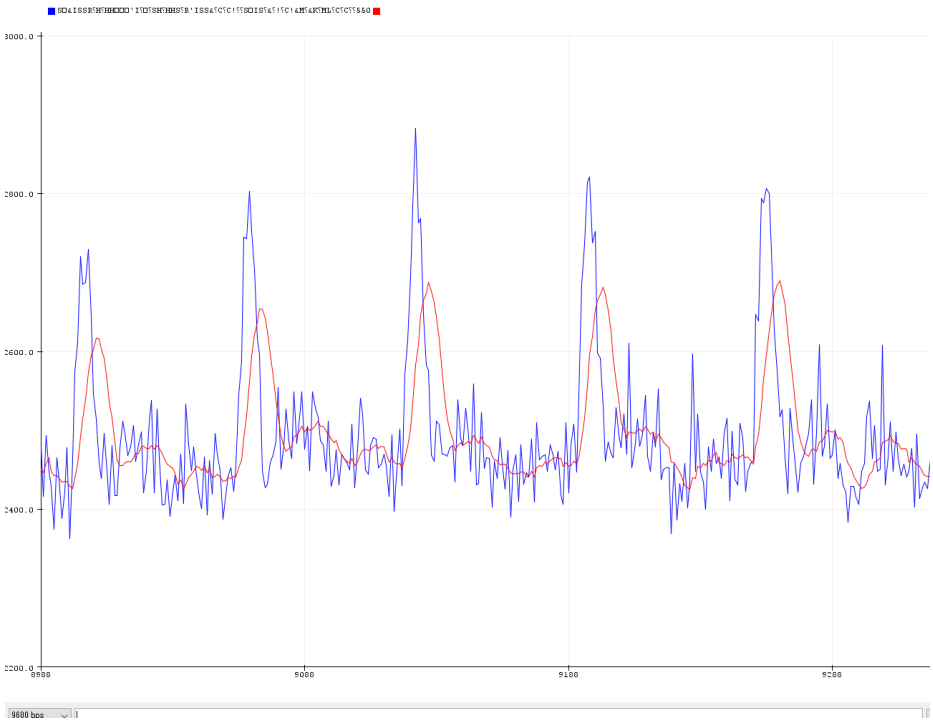

図 20 平滑化前と平滑化後の比較　青：平滑化前　赤：平滑化後

6.1.3　閾値の設定

心拍の波とノイズによる波の識別をするために平滑化したデータから閾値を設定し,閾値を超えたら心拍の波と判定する.閾値の設定はデータ 50 個ごとの最小値と最大値を使用して,波の中間を閾値とするようにした

平滑化したデータを配列に格納し,5 0個ごとの最大値,最小値を取得する.

最大値,最小値の取得後,閾値の設定をする Threshold 関数を呼んでいる.(図 21)

```
h1[count1] = ave1;
count1 = count1 + 1;
if (count1 == 50) {
  for (int j = 0; j < n; j++) {
    sensorValMax1 = max(h1[j], sensorValMax1);
    sensorValMin1 = min(h1[j], sensorValMin1);
  }
  Threshold1();
```

図 21　最大値,最小値の取得

ノイズや指を入れ始めは最大値と最小値の差が大幅に変化するため今回は最大値と最小値の差が 100 以上 1000 以下であれば,閾値の設定を行うことにしている.ノイズや指の入れ始めによる値と判断した場合は閾値にセンサーで取得できる最大の値である 4095 を渡すことで値が超えないようになっている.(図 22)

```
void Threshold1() {
  if (sensorValMax1 - sensorValMin1 >= 100 && sensorValMax1 - sensorValMin1 <= 1000) {
    //閾値の設定
    Threshold_v1 = ((sensorValMax1 - sensorValMin1) / 2) + sensorValMin1;
  } else {
    Threshold_v1 = 4095;
  }
}
```

図 22　閾値の設定

閾値を算出する式は最大値を Max, 最小値を Min として, 心拍の波の中間 T を次の式より算出し, 閾値とした.

$$\mathbf{T} = \big((\mathbf{Max} - \mathbf{Min}) \div \mathbf{2}\big) + \boldsymbol{Min}$$

6.1.4 波の検出

課題として,閾値の設定をしてもノイズの波も心拍の波と判定をしてしまうことがあった.そのため, 閾値の設定に加えて波の間隔にも閾値を設定する必要があった.人の心拍の波の間隔は,約 0.6～1.2 秒なので閾値も 0.6～1.2 秒に設定した.しかし,波の間隔 0.6～1.2 秒でない人の場合ノイズの波と判定され値が取得することができなかった.また,処理速度の関係で時間に多少のタイムラグが発生していた. そこでノイズの波と心拍の波を識別できる範囲の中で間隔に余裕を持たせることにした.今回は 1.0 秒から 3.0 秒の範囲に設定している(図 23)

```
int Threshold_t_slow = 1000;
int Threshold_t_fast = 3000;
```

図 23　波の間隔設定

平滑化した値が閾値を超えたら配列に格納する.格納した値の波の間隔がノイズの波と判定した場合はその値は消去する.(図 24)

```
//閾値が超えたかどうか
if (ave1 > Threshold_v1) {
  x[cnt1] = ave1;
  //何回閾値を超えたか回数をカウントアップする
  cnt1++;
  Time1 = millis() - old1;
  old1 = millis();
  //波の間隔の判定
  if (Time1 < Threshold_t_slow || Time1 > Threshold_t_fast) {
    cnt1--;
  }
```

図 24　波の検出

6.1.5　データの保存

閾値を超えた値を 3 個取得すると state という変数に 1 を渡す.

その後,閾値を超えた値を数える変数,配列の変数,最大値,最小値の初期化をする.(図 25)

```
  if (cnt1 == 3) {
    state1 = 1;
    cnt1 = 0;
  }
  count1 = 0;
  sensorValMax1 = 0;
  sensorValMin1 = 4095;
}
```

図 25　データの保存

6.1.6　類似度判定

センサー二つとも閾値を超えた値を 3 個取得すると類似度判定をする match 関数を呼ぶ.(図 26)

```
if (state1 == 1 && state2 == 1) {
  match();
}
```

図 26　類似度判定の呼び出し

match 関数では類似度算出を行う．類似度の算出にはユークリッド距離を使用した．ユークリッド距離は 2 点間の距離を算出する方法の一つで，2 点が直線的にどれだけ離れているのかを示している． ユークリッド距離は値が小さいほど類似度が高いことを意味する．ユークリッド距離は次の式より求められる．

$$\mathbf{d}(\mathbf{x},\mathbf{y}) = \sqrt{(\boldsymbol{x_1}-\boldsymbol{y_1})^2+\cdots+(\boldsymbol{x_n}-\boldsymbol{y_n})^2}$$

今回は２つの脈波の２点間の距離を使用した.閾値を超えた値を３個取得すると類似度判定に入るため今回の場合,データの個数は３である.(図 27)

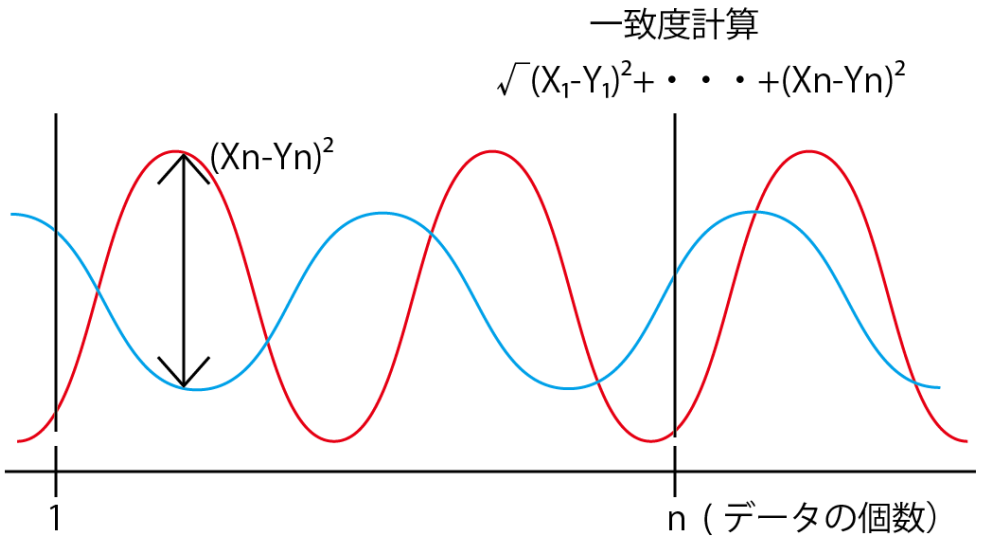

図 27 ユークリッド距離

算出された類似度によって場合分けをするための変数 var に値が 1 ~5 の段階に分けて,代入される.(図 28)

```
void match(void) {
  long sum = 0;
  //類似度の算出
  for (int j = 0; j < 3; j++) {
    long dev = pow((x[j] - y[j]), 2) / 100;
    sum += dev;
  }
  Euclid = sqrt(sum);

  if (Euclid < 80) {
    var = 1;
  } else if (Euclid < 100) {
    var = 2;
  } else if (Euclid < 150) {
    var = 3;
  } else if (Euclid < 200) {
    var = 4;
  } else {
    var = 5;
  }
```

図 28　類似度の算出

var に値が受け渡されると類似度に応じた光と音を出力させる関数を呼び出す.(図 29)

```
  switch (var) {
    case 1:
      rainbow();
      break;
    case 2:
      red();
      break;
    case 3:
      blue();
      break;
    case 4:
      green();
      break;
    case 5:
      purple();
      break;
  }
}
```

図 29　類似度の場合分け

6.2 初期化

類似度が決まって,それぞれの点灯と音の出力を 5 秒間行ったあと,init 関数で初期化をする(図 30)

```
void rainbow() {
  sound = 1;
  for (int i = 0; i < pixels.numPixels(); i++) {
    // ストリップの長さに沿って色相環（65536の範囲）を1回転させる量だけピクセルの色相をオフセットします。
    int pixelHue = step_num + (i * 65536L / pixels.numPixels());
    // ColorHSV関数に色相(0 to 65535)を渡し、その結果をgamma32()でガンマ補正します。
    pixels.setPixelColor(i, pixels.gamma32(pixels.ColorHSV(pixelHue)));
  }
  pixels.show();
  OSCMessage msg("/sens");            //OSCMessageの定義
  Udp.beginPacket(destIP, destPort);  //親機に対してUDPデータの書き込み開始
  msg.add(sound);                         //データの追加
  msg.send(Udp);                      //Pure Dataへデータ転送
  Udp.endPacket();                    //UDPデータの書き込み終了
  delay(5000);
  msg.empty();
  sound = 0;
  Udp.beginPacket(destIP, destPort);  //親機に対してUDPデータの書き込み開始
  msg.add(sound);                         //データの追加
  msg.send(Udp);                      //Pure Dataへデータ転送
  Udp.endPacket();                    //UDPデータの書き込み終了
  init();
}
```

図 30　初期化までの流れ

init 関数で Raw データが格納されている配列,無線通信で受け渡す音用の変数,閾値を超えた回数を数える変数,閾値を 3 回超えた状態を示す変数の初期化をし,点灯も取得中の色に変える.(図 31)

```
}
void init() {
  f[49] = { 0 };
  g[49] = { 0 };
  cnt1 = 0;
  cnt2 = 0;
  state1 = 0;
  state2 = 0;
  for (int c = step_num; c < pixels.numPixels(); c += 1) {
    pixels.setPixelColor(c, first);
  }
  pixels.show();
}
```

図 31　init 関数

LED と通信の処理については後ほど説明する.

6.3 LED 処理について

6.3.1　Adafurit_NeoPixel ライブラリ

LED の制御には Adafurit_NeoPixel ライブラリを使用した.（図 32）

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

図 32　Adafurit_NeoPixel ライブラリのインクルード

6.3.2　光の種類について

今回は５段階の類似度に応じた点灯と脈波を取得中の点灯,計６種類ある.

取得中はこのような点灯をしている.(**エラー! 参照元が見つかりません。**）

図 33　取得中の点灯

類似度に応じた点灯は高い方から虹,赤,青,緑,紫とした.(図 34)

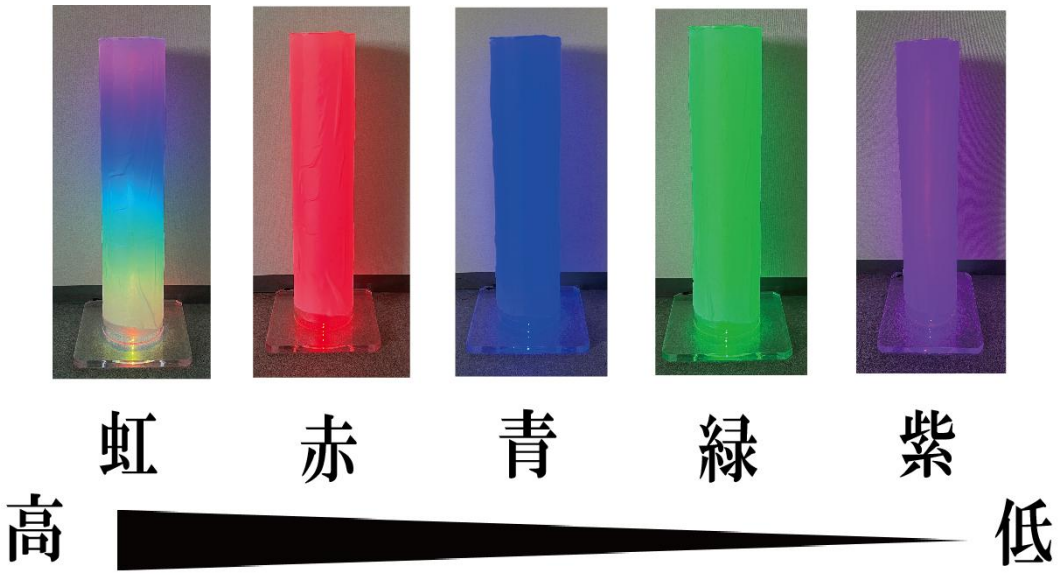

図 34　類似度に応じた点灯

6.3.3　色の処理について

(1)　虹について

今回使用する NeoPixel の性質を活かして LED 1 個ずつに違う色を出力させる処理にしている.

また,虹は制御が不安定で色にばらつきが出てしまうことがあったためガンマ補正の処理も入れている.(図 35)

```
void rainbow() {
  sound = 1;
  for (int i = 0; i < pixels.numPixels(); i++) {
    // ストリップの長さに沿って色相環（65536の範囲）を1回転させる量だけピクセルの色相をオフセットします。
    int pixelHue = step_num + (i * 65536L / pixels.numPixels());
    // ColorHSV関数に色相(0 to 65535)を渡し、その結果をgamma32()でガンマ補正します。
    pixels.setPixelColor(i, pixels.gamma32(pixels.ColorHSV(pixelHue)));
  }
  pixels.show();
```

図 35　虹色の処理

（2） 虹以外の処理について

虹以外の色の処理は RGB 値によって変化させている.(図 36)点灯の処理としては一つずつ RGB でしている値を受け渡すようになっている.(図 37)

```
int first = pixels.Color(255, 100, 255);  //桃
int similar2 = pixels.Color(255, 0, 0);   //赤
int similar3 = pixels.Color(0, 0, 255);   //青
int similar4 = pixels.Color(0, 255, 0);   //緑
int similar5 = pixels.Color(255, 0, 255); //紫
```

図 36 色の指定

```
void red() {
  sound = 2;
  for (int c = step_num; c < pixels.numPixels(); c += 1) {
    pixels.setPixelColor(c, similar2);
  }
  pixels.show();
```

図 37　点灯の処理

6.4 Pure Data(以下 Pd)

6.4.1　ライブラリ

今回の研究では,音の出力は無線通信を使用することにした.ライブラリは OSC と WiFi を使用した.(図 38)

```
#include <OSCBoards.h>
#include <OSCBundle.h>
#include <OSCData.h>
#include <OSCMatch.h>
#include <OSCMessage.h>
#include <OSCTiming.h>
#include <SLIPEncodedSerial.h>
#include <SLIPEncodedUSBSerial.h>

#include <ETH.h>
#include <WiFi.h>
#include <WiFiAP.h>
#include <WiFiClient.h>
#include <WiFiGeneric.h>
#include <WiFiMulti.h>
#include <WiFiSTA.h>
#include <WiFiScan.h>
#include <WiFiServer.h>
#include <WiFiType.h>
#include <WiFiUdp.h>
```

図 38　ライブラリ

6.4.2　　変数

通信用に用いた変数については以下のとおりである.(図 39)

```
const char* ssid = "*";         //WiFiのssid
const char* password = "*";  //WiFiのパスワード
unsigned int localPort = 8700;         //自身のポート番号
byte destIP[] = { 172, 16, 64, 64 };  //相手PCのIPアドレス
int destPort = 9001;                   //Pure Dataで受信するUDPのポート番号
WiFiUDP Udp;                           //UDP
int sound = 0;                         //Pure Data用
```

図 39　通信用の変数(ssid と password は非表示)

・ssid：Wi-Fi の SSID

・password：Wi-Fi のパスワード

・localPort：ボード側のポート番号

・destIP[]：相手側の IP アドレス

・destPort：Pure Data で受信する UDP のポート番号

・Udp：UDP 通信するための変数

・sound：Pure Data 用変数

6.4.3　Wi-Fi と UDP 接続

まず Wi-Fi 接続を開始する.Wi-Fi 接続されていない間,シリアルモニタに”.”表示する.Wi-Fi 接続に成功したら,シリアルモニタに”Wi-Fi connected.”が表示させる.その後,UDP 接続を開始する.(図 40)

```
Serial.begin(9600);
WiFi.begin(ssid, password);  //WiFi接続開始

/*WiFi接続されているかどうかの判定*/
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
}

/*WiFi接続成功時の確認*/
Serial.println();
Serial.println("WiFi connected.");

Udp.begin(localPort);  //UDP接続開始
```

図 40　Wi-Fi と UDP 接続

6.4.4　OSC 通信

OSC 通信については以下のとおりである.(図 41)

・OSC Message 型の変数 msg(“/sens”)を定義する.

・UDP データの書き込みを開始する.

・Pure Data 用の変数 sound を msg に追加する.

・データを Pure Data へ転送する.

・書き込みを終了する.

```
Udp.beginPacket(destIP, destPort);   //親機に対してUDPデータの書き込み開始
msg.add(sound);                      //データの追加
msg.send(Udp);                       //Pure Dataへデータ転送
Udp.endPacket();                     //UDPデータの書き込み終了
```

図 41　OSC 通信

6.4.5　機能

Pd ファイルやプログラムのことをパッチ(patch)と呼ぶ.Pd.プログラミングは,オブジェクトと呼ばれる各々異なる機能を持った[箱]を,パッチコードと呼ばれる[線]で繋いでいくことが基本となる.

6.4.6　主なオブジェクト

Pd には様々な種類のオブジェクトが存在する．ここでは主なオブジェクトの説明をする．(図 42)

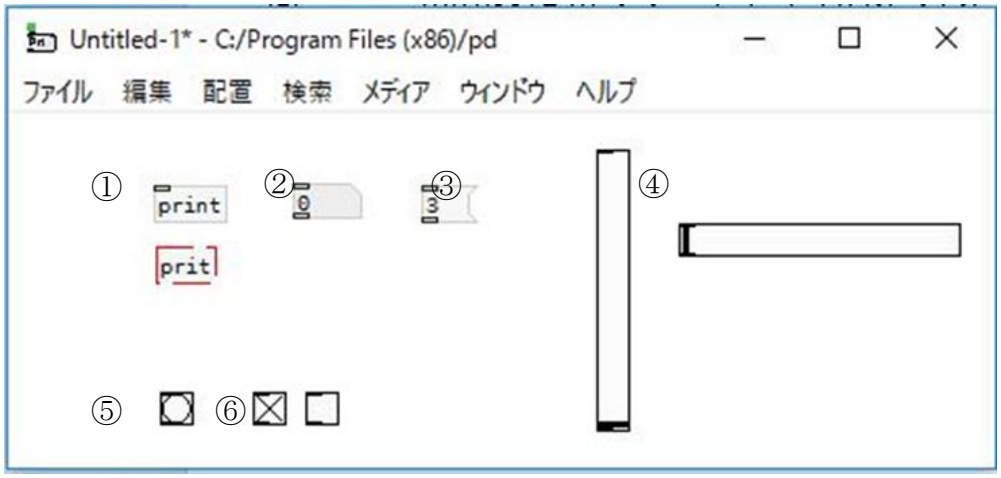

図 42　様々なオブジェクト

①オブジェクト：オブジェクト名を入力して設定する箱．オブジェクト名が間違っていると赤い点線になる．

②ナンバーボックス：数値データを出力する．

③メッセージボックス：数値や文字列データ，リストを出力する．

④スライダー（垂直・水平）：設定した範囲内で数値の出力をする．

⑤Bang ボタン：実行モードでクリックすると Bang データを出力する．

⑥Toggle ボタン：チェックのオン/オフで数値データの 1/0 を出力する．

6.4.7　　インレット・アウトレット

オブジェクトの上についている「穴」をインレット,下についている「穴」をアウトレットという.インレットはメッセージ,すなわちデータやオブジェクトに対する命令を入力するために使う.また,アウトレットからはオブジェクトの行った処理の結果が出力され,それがほかのオブジェクトへのメッセージとなる.

インレット,アウトレットは複数ある場合が多いため,それぞれに名前がある.インレットが２つの場合は,左インレット,右インレットと呼ぶ.また３つの場合は,左インレット,中央インレット,右インレットと呼ぶ.まれに,それ以上ある場合は左から第１インレット,第２インレット,第３インレット……と呼ぶ.(図 43)

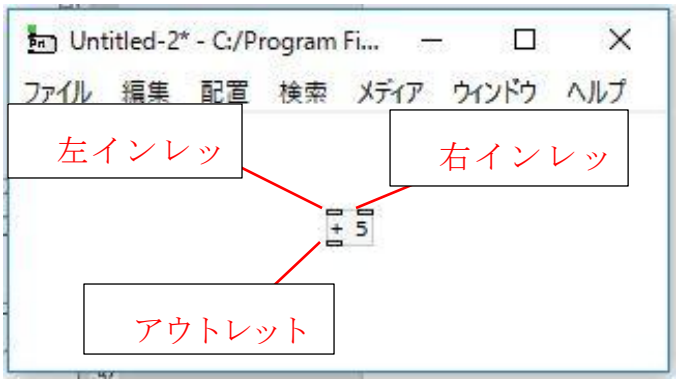

図 43 インレット・アウトレット

6.4.8　パッチの作成方法

作成方法について簡単なパッチを用いて説明する．まず，Pd を起動すると，Pd ウィンドウが開かれる．Pd ウィンドウは Pd 全体の機能設定や，動作ログやエラーメッセージを表示するためのものである．ウィンドウを閉じると Pd 自体が終了してしまう．（図 44）

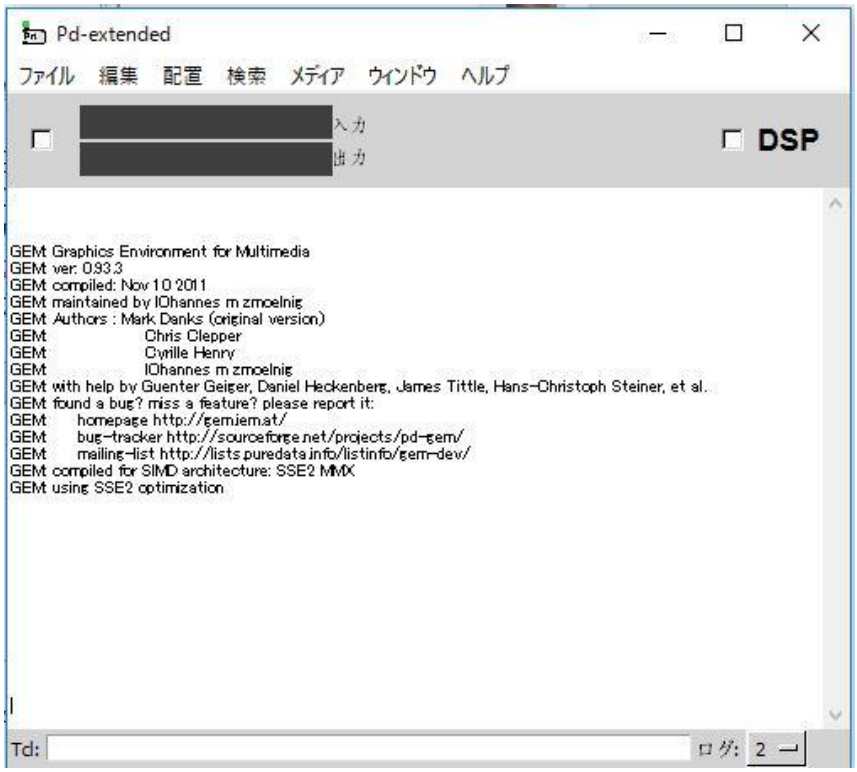

図 44　　Pd ウィンドウ

Pd のプログラムは，キャンバスと呼ばれる白地のウィンドウ上に，さまざまな箱を配置し，それらを境界線でつなぐことでデータの流れをプログラムする．ここではサンプルとして，Hello world と表示するパッチを作成していく．

配置メニューからメッセージボックスを選択し適当な位置に配置する．メッセージボックス内でカーソルが点滅するため，Hello world（以下[Hello world]）と入力する．Pd に入力が終了したことを知らせるためにキャンバスの白地の部分をクリックする．

次に配置メニューからオブジェクトを選択し適当な位置に配置する．オブジェクト内には print（以下[print]）と入力し，キャンバスの白地部分をクリックする．（図 45）

図 45 sample.pd（1）

最後にマウスポインタを[Hello world]の左下にある小さな四角（アウトレット）に合わせクリックし，[print]の左上にある小さな四角（インレット）までドラッグする．マウスボタンを離せば，[Hello world]から[print]へのパッチコードが生成される．（図 46）

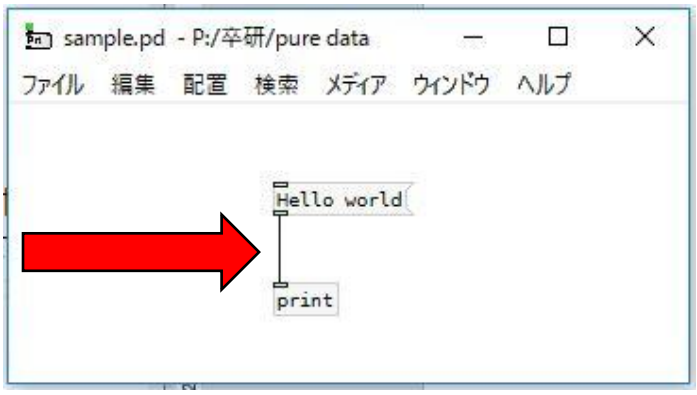

図 46　sample.pd（2）

プログラムを実行するには編集メニューから編集モードを選択し，実行モードに変更する（編集モードの場合カーソルは手のマークになり，実行モードの場合は矢印に戻る）．実行モードにした状態で，

[Hello world]をクリックすると，Pd ウィンドウのコンソールに print: Hello world という行が追加される．（図 47）

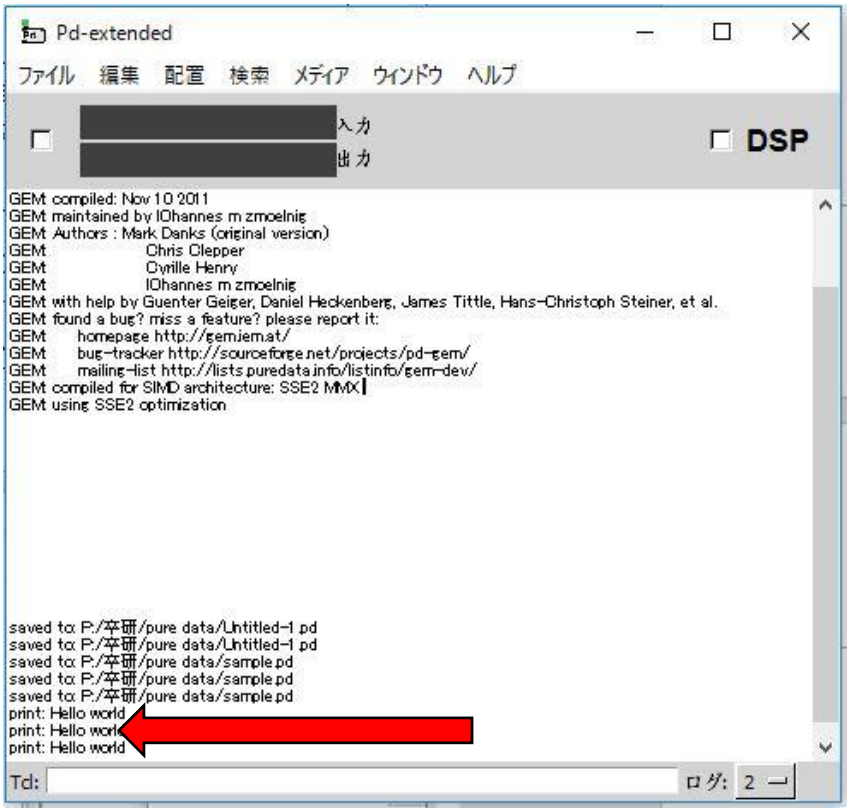

図 47　実行結果

6.4.9　　受信と音の出力

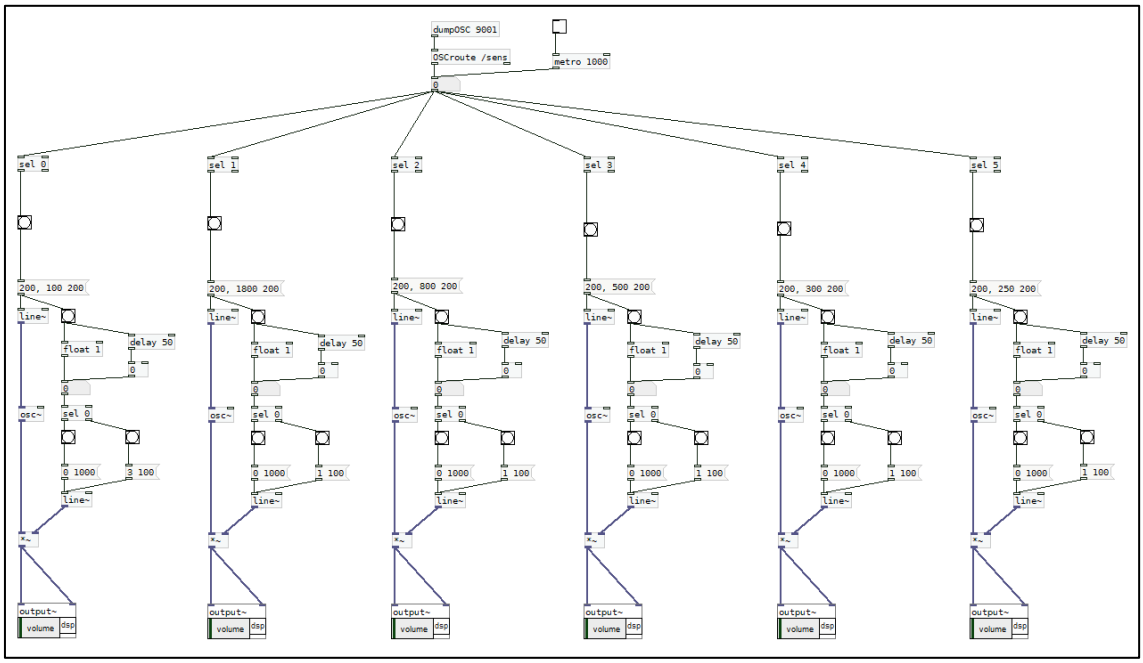

図 48 OscSound.pd パッチ

(1)　オブジェクト

・[dumpOSC]：指定したポート番号で OSC データを受信する.

・[OSCroute]：受信したデータの振り分けを行う.

・[metro]：一定間隔で bang を出力する.

・[sel]：インレットからの入力値と引数を比較する.入力値と引数のいずれかが一致する場合は対応するアウトレットから bang を出力し,一致しない場合は 1 番右のアウトレットから入力値をそのまま出力する.

・[line~]：入力された float 値を signal に変換.所定の時間をかけて徐々に signal の値を上昇,下降させるときにも使用する.

・[float]：float 値を保存し,左インレットに bang を入力された時にその値を出力する.

・[osc~]：周波数を指定して,正弦波を出力する

・[*~]：左インレットに入力された signal の値 a と右インレットに入力された引数の値 b を乗算し,その結果を出力する

(2)　パッチ

このパッチに送られてくる値は,0~6 のどれかである.測定中の時には,常に 0 が送られる.そして判定したら,1~5 が送られてくる.1~5 には類似度判定で出た光り方に対応した音が出る.類似度が高くなるにつれて高い周波数の音が出る.大まかに説明したところで,次にパッチの詳細について部分的に説明していく.

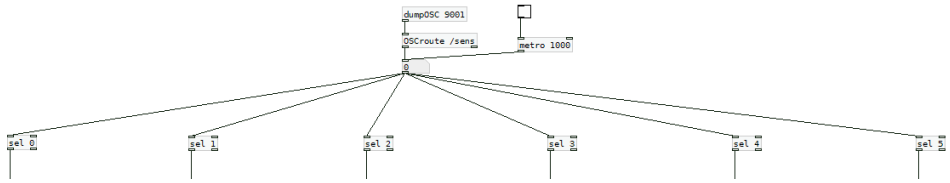

図 49　　OSC データの受信

ここでは,OSC 通信の中の,データの受信についての処理を行っている.まず,dumpOSC オブジェクトで指定したポート番号の OSC データを受信する.次に,OSCroute オブジェクトで受信したデータの振り分けを行う.プログラム内では,/sens という変数に値が振り分けられている.そして,sel オブジェクトで/sens の値が指定した数値なら,処理を行う.

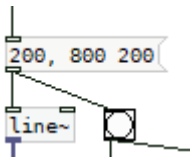

図 50　　周波数変化

ここでは,出力する音の周波数の変化させる処理を行っている.line オブジェクトでメッセージボックスの第 1 要素の 200hz から第 2 要素の 800hz へ第 3 要素の 200ms をかけて徐々に変化させる.

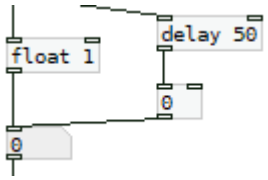

図 51　　音量変化

ここでは,音量を変化させる処理を行っている.float オブジェクトで指定した数値 1 をナンバーボックスに出力している.それで,500ms たったら, 0 をナンバーボックスに出力している.

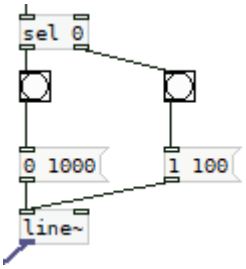

図 52　　ノイズを防ぐ処理

続いての説明に入る.前図の処理だけで音量を変えることができるが,音の再生時と停止時に「ブチッ」というノイズが発生してしまう.よって,それを防ぐ処理を行っている.

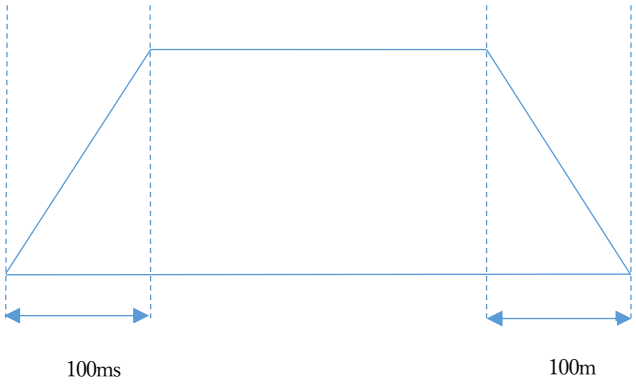

図 53　ノイズを防ぐ方法

ノイズを防ぐ方法は,音量をいきなり変化させるのではなく,徐々に変化させることである.

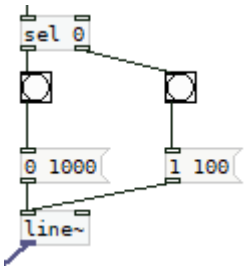

図 54　　ノイズを防ぐ処理

まず,先ほど音量の説明ででたナンバーボックスの値によって,処理が変わる.

値が 1 の時,sel オブジェクトの右側の処理を行い,line~オブジェクトで値を徐々に変化させていく処理を行うことで音量を上げていく.値が 0 の時,sel オブジェクトの左側の処理を行い,line~オブジェクトで値を徐々に変化させていく処理を行うことで音量を下げていく.

(3)　音の出力方法

Pd で音の出力を行うには，output~オブジェクトの dsp が ON になっている必要がある．dsp をクリックすることで背景が緑色になり×マークがつく.この状態で volume のバーを動かすことで音量の調整ができる.

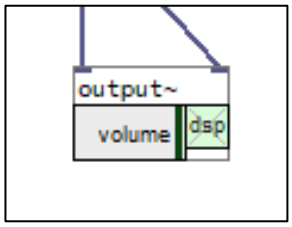

図 55　　output~オブジェクト

6.4.10　音の種類について

音も類似度に応じた周波数と脈波を取得中の周波数,計６種類ある.

取得中は 100Hz の音を出している.

類似度に応じた周波数は高い方から 1800Hz,800Hz,500Hz,300Hz,250Hz とした.(図 56)

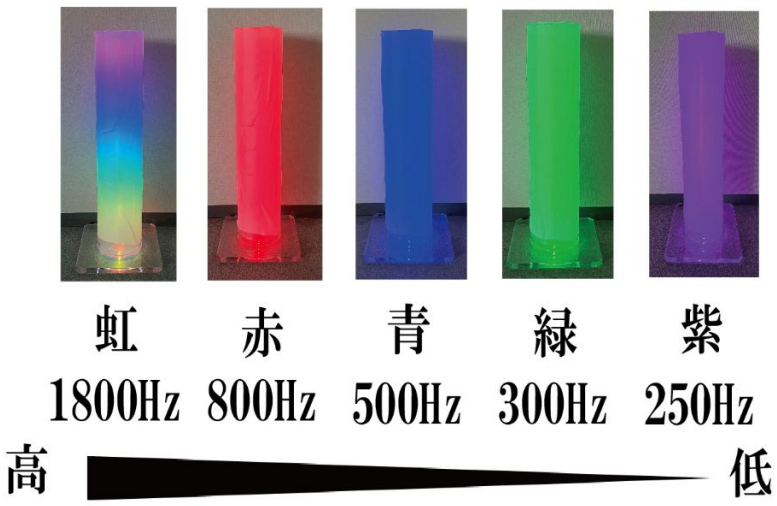

図 56　類似度に応じた周波数

第7章　終わりに

今回は昨年の研究の引継ぎをし,展示をできる状態を目指し改善と機能拡張を行った.昨年の研究で見つかった接続の問題点を早期に改善することができた.

一方で,コードの修復は報告書に説明がかいてあるにもかかわらず理解・修復に時間を多く費やしてしまい,力不足も実感した.

しかし,機能拡張として音の機能を追加し,体験したひとがより楽しめるアート作品を作成することができた.

「水と向き合う」のようにこれからの産技短のイベントで展示され,実際に触れ,見て楽しんでもらえたらうれしい.

参考文献

[1]入力デバイスに水を用いて音を生成するアート作品(産業技術短期大学校矢巾校卒業研究報告書,2016,上山明江,東山真実)

[2] 入力デバイスに水を用いて音を出力・制御する(産業技術短期大学校矢巾校卒業研究報告書,2017,廣藤美緒)

[3] 入力デバイスに水を用いて音と光を出力するシステムの制作(産業技術短期大学校矢巾校卒業研究報告書,2018,田鎖鴻容)

[4] 自由と独立を意味する［independants］を冠した無審査の美術展。仙台から。つづけ。アート。ゴー。

https://sendai21-independants.com/#gallery

[5] 「アート＆テクノロジー東北２０１８」コンテスト

http://www-cg.cis.iwate-u.ac.jp/AT2018/index.html

[6]心拍信号を利用したアート作品の制作(産業技術短期大学校矢巾校卒業研究報告書,2022,橋本佳太,福田陸斗)

[7]「水と向き合う」

http://www-cg.cis.iwate-u.ac.jp/AT2018/award.html

[8]ESP32 と Neo Pixel フルカラーLED テープで Wi-Fi 卓上イルミネーションオブジェを作ってみた

https://www.mgo-tec.com/blog-entry-led-tape-neopixel-esp32-artnet.html/2

[9]第 30 回 OSC 通信で Arduino と他のアプリを連携させてみる.(前編)

https://deviceplus.jp/raspberrypi/entry0030/

[10]Pure Data Japan　日本の Pure Data ユーザのためのフォーラム

https://puredatajapan.info/?page_id=2

[11]garretlab　いろいろ試したことを書き留めるページです

https://garretlab.web.fc2.com/

[12]Pure Data　チュートリアル＆リファレンス(美山千香氏,株式会社ワークスコーポレーション,2013)

[13]Pd Recipe Book ―― Pure Data ではじめるサウンドプログラミング(松村誠一郎,株式会社ビー・エヌ・エヌ新社,2012)

[14]UDP とは

https://www.nic.ad.jp/ja/basics/terms/udp.html