関西国際大学研究紀要　第 24 号，2023 年，211－220

Arduino モーションセンサーによる MIDI コントローラの製作

Making a MIDI Controller with Arduino Motion Sensor

岡　本　　久*
Hisashi OKAMOTO

Abstract

Microcomputers are becoming smaller and smaller as well as more powerful and multifunctional, and they are incorporated into all kinds of devices in our daily lives. In the field of education, microcomputer boards such as PIC, Arduino, and Raspberry-Pi are widely used, and a wide variety of sensors and motor controllers are commercially available. By combining these devices and programming them with dedicated libraries, it is possible to create highly accurate and practical products in a relatively short time.

Arduino is a project that focuses on allowing non-technical students to learn practical control programs, and is an integrated system environment of hardware and software that can be easily expanded by stacking boards equipped with sensors and other devices called "shields" on the main board mounted with an AVR microcontroller. This is an integrated system environment of hardware and software that can be easily expanded.

This paper is a report on a simple MIDI controller from the author's research on computer music devices, which can be used for control programming education.

キーワード：Arduino, Motion Sensor, MIDI Controller, モーションセンサー, MIDIコントローラ

Ⅰ. はじめに

マイクロコンピュータは高性能・多機能化とともに小型化が進み，日常生活のあらゆる機器に組み込まれている。教育の分野でもPICやArduino，Raspberry-Piなどのマイコンボードが幅広く利用され，センサーやモーター制御装置など多種多様なものが市販されており，それらを組み合わせて専用ライブラリを利用しプログラミングを行えば，比較的短期間で精度の高い実用的なものを作ることができるようになっている。

Arduino　は技術系ではない学生でも実践的な制御プログラムを学ぶことができることに重点をおいたプロジェクトであり，AVR　系マイコンを実装した本体ボードに「シールド」と呼ばれるセンサーなどを搭載したボードを積み重ねることで，容易に機能拡張が行えるハードウェアとソフトウェアの統合システム環境である。

本稿は筆者が行っているコンピュータ音楽系装置の製作研究の中から，制御プログラミング教育などでも活用できることを考えた，シンプルなMIDIコントローラについてのレポートである。

*関西国際大学 社会学部

Ⅱ．モーションセンサーの操作よる MIDI コントロール

１．Arduino ボードとシールドの構成

モーションセンサーは角度や加速度などを測定するためのセンサーで，携帯電話やドローンなど様々な機器に組み込まれている。本稿ではこのセンサーを用い，操作に従い MIDI 音源の発音などをコントロールする装置について紹介する。具体的には Arduino マイコンボードに 9 軸センサーシールド（3 軸加速度・3 軸ジャイロ・3 軸地磁気）と MIDI 音源シールドをマウントし，センサーの動きに合わせて MIDI メッセージを生成，発音・変化させる。使用したマイコンボードおよびシールドは下記のとおりである。

- Arduino マイコンボード　　Arduino UNO R3（AVR ATmega328P）
- 9 軸モーションシールド　　Arduino 9 axis motion shield （BOSCH BNO055）
- MIDI 音源シールド　　sparkfun Musical Instrument Shield（VLSI VS1053）

２．ボード、シールドの写真

ボード，シールドの写真を図 1 に示す（左から Arduino マイコンボード，９軸モーションシールド，MIDI 音源シールド）。またこれらのボードを実際に重ね合わせた状態は図 2 のとおりである。これらのボードやシールドは，すでに発売から 10 年以上経過した古いものであるが，シンプルで扱いやすく，今回の製作には十分な性能を備えている。

図 1. Arduino マイコンボードと 2 つのシールド

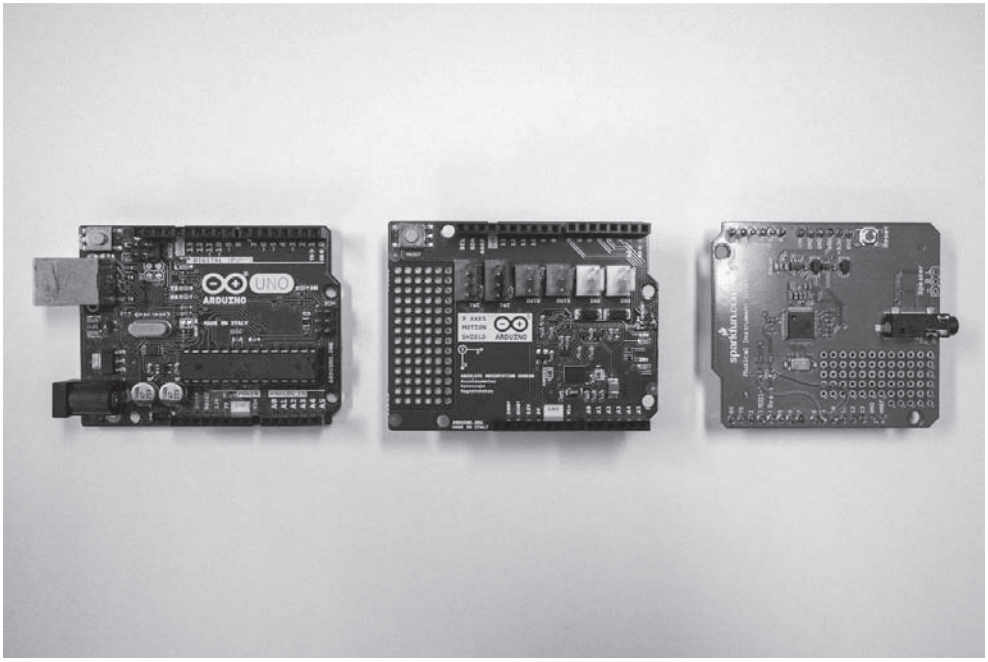
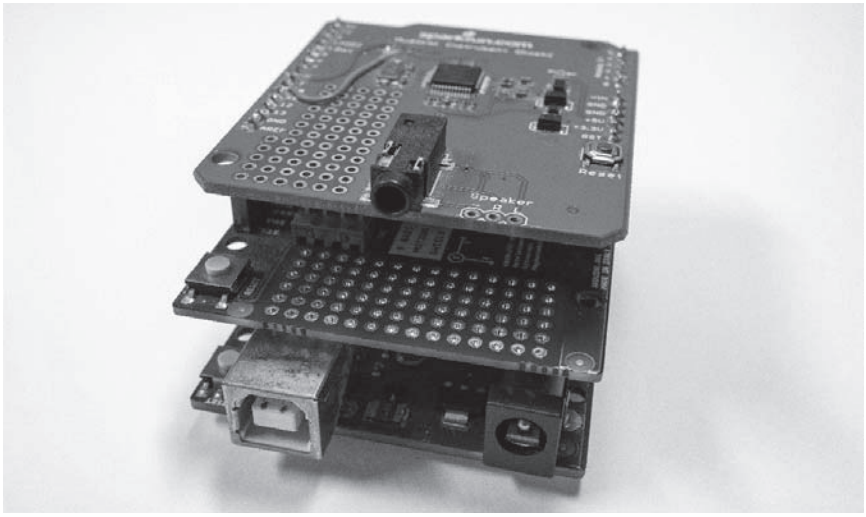


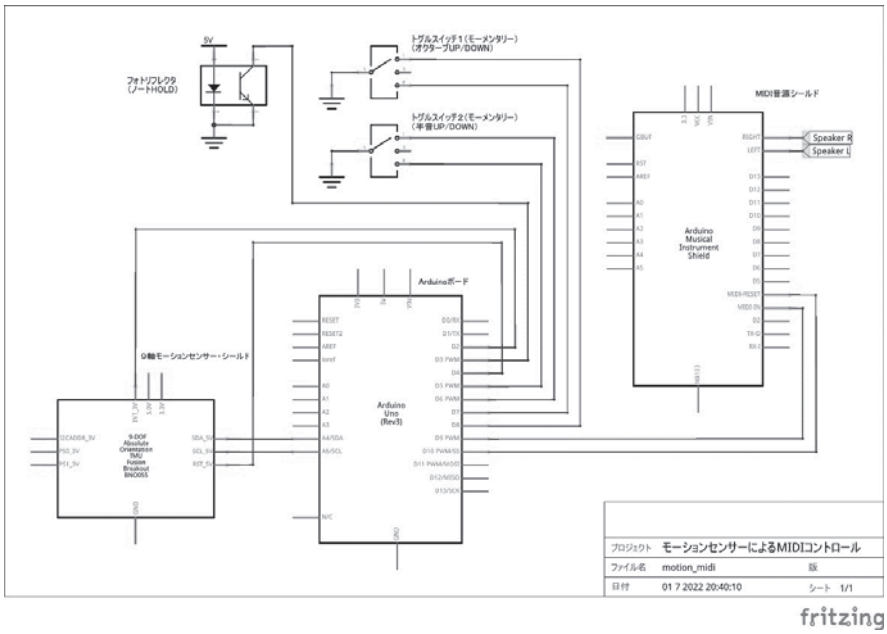
図 2. Arduino マイコンボードに 2 つのシールドを重ねた状態



３．回路図

回路図を図 3 に示す。簡略化のため電源や基本的な結線，プルアップ抵抗などは省略している。また図上にはトグルスイッチやフォトリフレクタが存在しているが，本稿執筆時点ではプログラム上でのシミュレーションのみ行っている。

図 3. 回路図



４．操作と MIDI メッセージの関係

操作は図 2 のモジュール全体を持ち上げ，前後左右に移動や回転をさせる。写真にはないが，実際には Arduino マイコンボードに USB ケーブルを接続し電源を供給，MIDI 音源シールドにはイヤホンをつないで音を確認している。

操作が行われると Arduino マイコンボードはモーションセンサーの値を読み取り，所定の MIDI メッセージを生成，MIDI 音源シールドに送信する。生成する MIDI メッセージはノート ON，ピッチベンド，オールノート OFF のいずれかに限定している。操作と MIDI メッセージの関係を表 1 に示す。

５．プログラム作成と動作確認

プログラム作成のため，パソコンと Arduino ボードを USB ケーブルで接続。専用の Arduino IDE 上でコーディングし，Arduino 本体に書き込んで動作確認を行う。なお，センサーの変化に対する MIDI メッセージ生成の手順は表 2 の通りである。また今回のプログラムで鳴らす音の種類（音色）は，時間経過とともに自然に音が消えるピアノやビブラフォンなど減衰音のみとしており，ノート OFF の処理は行っていない。

表 1. 操作と MIDI メッセージの関係

操作	方法	MIDI メッセージ
音程選択	左右に傾ける（Roll 角操作）	Roll 角に従いハ長調音階 B3～D5（10 音）のいずれかが選択される
オクターブ変化	オクターブ・スイッチ操作	スイッチの状態によりオクターブ UP または DOWN する
半音変化	セミトーン・スイッチ操作	スイッチの状態により半音 UP または DOWN する
音程をホールド	ノート HOLD ボタン操作	押下中は現在選択中の音程が固定される
発音（音を出す）	振り下ろす（Pitch 角で上から下）	Pitch 角がマイナスからプラスになると選択中の音が発音（ノート ON）される
発音時の強さ	振り下ろすときの速度を変える（Z 軸）	振り下ろす速度に応じ発音時のベロシティ値が変化する
ビブラート	左右に揺らす（X 軸）	ピッチベンド・メッセージによりビブラート効果が生じる
発音停止	前後±80 度以上傾ける（Pitch 角）	オールノートオフ・メッセージによりすべての音が停止する

表 2. センサー入力と MIDI メッセージ出力との関係

入力デバイス	検出対象	読み取り値	生成する MIDI メッセージ	仕様
9 軸モーションセンサー	オイラー角	Roll 角の値	ノートナンバー	±45 度範囲を 10 分割しノートナンバー60〜74（B3〜D5）のハ長調音階（10 音）にアサインする
		Pitch 角の値	ノート ON メッセージ	マイナス角からプラス角に変化したときノート ON 処理を行う
			オールノート OFF メッセージ	±80 度を超えたときオールノート OFF 処理を行う
	加速度	X 方向の加速度	ピッチベンド・メッセージ	±1/2 半音の範囲でビブラート効果
		Z 方向の加速度	ベロシティ値	加速度に応じベロシティ値を算出
オクターブ・トグルスイッチ ON-OFF-ON（モーメンタリー）	ON-OFF-ON の位置	オクターブ UP の状態	ノートナンバーを 1 オクターブ UP	ON で音程をオクターブ UP
		オクターブ DOWN の状態	ノートナンバーを 1 オクターブ DOWN	ON で音程をオクターブ DOWN
セミトーン・トグルスイッチ ON-OFF-ON（モーメンタリー）	ON-OFF-ON の位置	半音 UP の状態	ノートナンバーを半音 UP	ON で音程を半音 UP
		半音 DOWN の状態	ノートナンバーを半音 DOWN	ON で音程を半音 DOWN
フォトリフレクター	ON または OFF	ノート HOLD の状態	現在のノートナンバーを HOLD	ON で選択中の音程を HOLD

６．動作確認の結果

動作確認の結果，おおむね期待通りの結果を得ることができた。しかしながら下記のような大きな問題も確認された。[注1]

- Roll 方向の傾きだけで，どの音程が選択されたのかを正確に判断することは困難。
- Roll 方向の傾きを維持したまま振り下ろしても，ブレにより音程が変わってしまう。そのためノート HOLD 機能を設けているが，操作がやや複雑になってしまう。
- 振り下ろしても音が鳴らないときがある。これはモーションセンサーが一定速度以上の変化に追従できていないためであると考えられる。
- 振り下ろした速度に応じたベロシティにならないときがある。これも上記同様の理由と考えられる。
- 上記４点の問題もあり，速いテンポで演奏することができない。

Ⅱ．プロトタイプ機の製作

１．プロトタイプ製作に先立つ留意点

今回「基盤むき出し」による実験レベルで一定の結果は出たものの，プロトタイプ機としてケースなどに収めるとセンサーの位置が変わることで感度などが変化し，実験とは異なる結果になることが少なくない。そのためにも各種スイッチなどすべてを実装，適切な位置に配置した形でのプロトタイプ機を製作し，既知の問題を含め改めて検証していく必要がある。

２．入出力を２つのモジュールに分ける

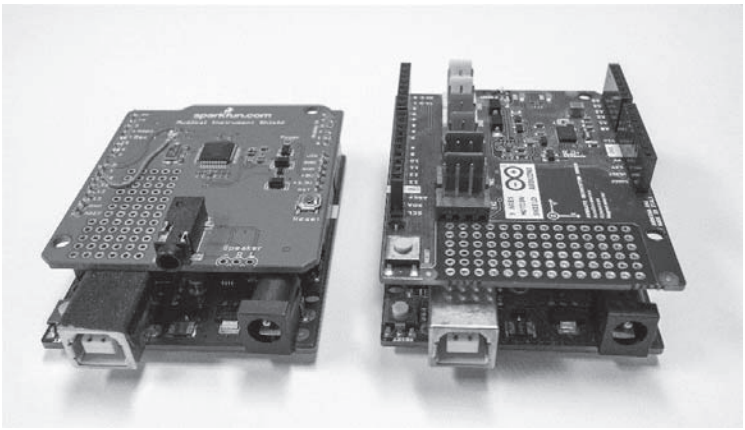
上記では Arduino の本体ボードにセンサーボードと音源ボードを重ね，ひとつのモジュールの状態で動作確認を行った。しかしプロトタイプ機では操作性やケーブルの引き回しなどの実用面や，MIDI 音源側の処理をさらに展開していくことを計画しているため，操作部と音源部とを２つに分けることを予定している。そこでプロトタイプ機製作に先立ち，モジュールを２つに分けて簡単なテストを行った。

３．「センサー側 Arduino」と「音源側 Arduino」

図４の写真のように Arduino のマイコンボードを２枚用意し，１枚目にモーションセンサーを載せ「センサー側 Arduino」，２枚目に MIDI 音源シールドを載せ「音源側 Arduino」としてモジュールを分けた。またテストではモジュール間の送受信データを確認するため，Arduino 同士を直接ではなく，それぞれパソコンと USB シリアルで接続し，センサー側からの受信データを音源側 Arduino に転送する形で行った。

- センサー側 Arduino は，モーションセンサーの値から MIDI メッセージを生成し PC 上に送信。
- PC 上のソフト（Processing）でデータを受信し，そのまま音源用 Arduino に転送。またデータ内容をモニターに表示。
- 音源側 Arduino は，受信した MIDI メッセージを MIDI 音源ボードにそのまま送信。

図４．「センサー側 Arduino」と「音源側 Arduino」の２つのモジュール



４．回路図およびプログラム

この２つのモジュールに分けた回路図を図５に示す。またセンサー側 Arduino および音源側 Arduino のプログラムをそれぞれ図 6，図 7 に示す。それぞれのプログラムは主要な部分のみを掲載している。この実験の結果については、２段階になった通信の影響により操作から発音までのレイテンシーがわずかに大きくなったようにも感じられたものの，大きな問題は出なかった。むろん速い操作に対し追従が伴わない問題はそのままであるため，今後詳しく原因を調べるとともに対策を立てたい。

図 5. ２つのモジュールに分けた回路図

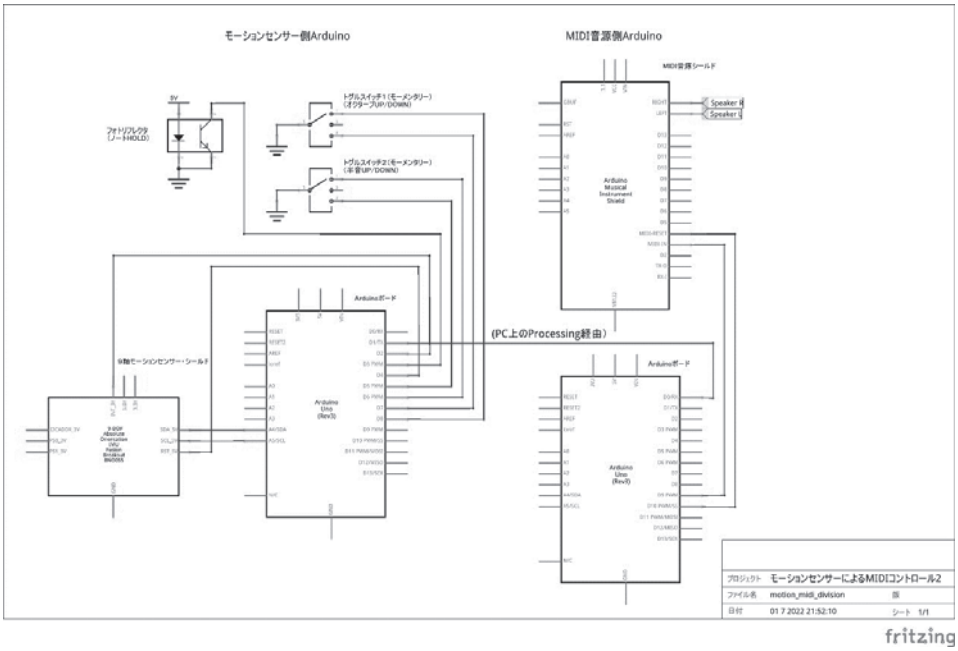


図 6. センサー側 Arduino のプログラム

```
void loop()
{
  float accelX;                  // 加速度 X 軸
  float accelZ;                  // 加速度 Z 軸
  float eularPitch;              // オイラー・ピッチ角
  float eularRoll;               // オイラー・ロール角

  /* 9 軸センサーのアップデート */
  if (updateSensorData)                          //センサーのアップデート・フラグが TRUE の場合は処理を行う
  {
    mySensor.updateAccel();                      // Update the Accelerometer data
    mySensor.updateEuler();                      // Update the Euler data into the structure of the object
    mySensor.updateCalibStatus();                // Update the Calibration Status
    updateSensorData = false;                    // センサーのアップデート・フラグを FALSE にリセット
  }

  /* 一定時間間隔でのセンサー読み込み処理 */
  if ((millis() - lastStreamTime) >= streamPeriod)      // 所定のインターバルタイムが経過した場合は処理を行う
  {
    lastStreamTime = millis();                   // 現在タイムを直前タイムに保存
    accelX = mySensor.readAccelerometer(X_AXIS);     // X 軸の加速度を得る
    accelZ = mySensor.readAccelerometer(Z_AXIS);     // Z 軸の加速度を得る
    eularRoll = mySensor.readEulerRoll();            // オイラーの Roll 角を得る
    eularPitch = mySensor.readEulerPitch();          // オイラーの Pitch 角を得る
    perfome_accelX(accelX);                          // 加速度センサーの X 軸の処理（ピッチベンドによるビブラート効果）
    perfome_eular(eularRoll, eularPitch, accelZ);    // オイラー角による処理（ノート発音制御）
    updateSensorData = true;                         // センサーのアップデート・フラグを TRUE にする
  }
}


/*************************************************************************
  加速度センサーの X 軸操作時の処理：ピッチベンドによるビブラート効果
*************************************************************************
*/
void perfome_accelX(float accelX)
{
  int16_t bender;                // MIDI ピッチベンド値（14bit）
  static int16_t lastBender = 0;     // 直前のピッチベンド値（初期値はセンター）

  bender = (int16_t)map(constrain(accelX, -40.0, 40.0), -40.0, 40.0, -BENDER_MAX, BENDER_MAX); // 加速度センサーの振れ幅（実測値）をピッチベンドの振れ幅（±抑制値）
に変換し送信
  if (bender != lastBender)          // 直前のピッチベンド値と異なるとき
  {
    txMIDI_pitchbend(0, bender);     // ピッチベンド・メッセージを送信
    lastBender = bender;             // 直前のピッチベンド値を更新
  }
}


/*************************************************************************
  発音処理：センサーの傾きやボタン押下状況によりノート、ベロシティを算出し MIDI 出力する
*************************************************************************
*/
void perfome_eular(float roll, float pitch, float accel)
{
  uint8_t num;                   // MIDI ノートデータの配列番号
  uint8_t note;                  // ノートナンバー
  uint8_t vel;                   // ベロシティ
  static uint8_t lastNote = 60;  // 直前のノートナンバー
  static float lastEularPitch = 0;  // 直前のオイラー・ピッチ角


  if (digitalRead(PIN_NOTE_HOLD) != LOW) {       // ノート HOLD ボタンが押下中でないとき

    // Roll 角をもとに音程決定のためのデータを算出
    num = (uint8_t)map(constrain(roll, -45.0, 45.0), -45.0, 45.0, 9.0, 0.0);    //±45 度のロール値を 10 段階に区分する
    note = gc_note[num];             // num の値をもとにノートナンバー配列からノートナンバーを得る

    // オクターブ・アップ／ダウンボタンの状態による処理
    if (digitalRead(PIN_OCTAVE_UP) == LOW) note += 12;          // オクターブ・アップボタンの押下時
    else if (digitalRead(PIN_OCTAVE_DOWN) == LOW) note -= 12;   // オクターブ・ダウンボタンの押下時

    // 半音 UP/DOWN ボタンの状態による処理
    if (digitalRead(PIN_SEMITONE_UP) == LOW) note++;            // 半音 UP ボタン押下時
    else if (digitalRead(PIN_SEMITONE_DOWN) == LOW) note--;     // 半音 DOWN ボタン押下時
  }
  else {                         // ノート HOLD ボタン押下中のとき
    note = lastNote;               // lastNote の値を note の値とする
  }
```

```
  // Z 方向の加速度からベロシティ値を算出
  vel = (uint8_t)map(constrain(accel, ACCEL_Z_LOWER, ACCEL_Z_UPPER), ACCEL_Z_LOWER, ACCEL_Z_UPPER, VELOCITY_LOWER, VELOCITY_UPPER);

  // Processing 向けのノート情報送信
  if (note != lastNote)                    // 直前のノートナンバーと異なるとき
  {
    txMIDI(0xB0, 0x50, lastNote);          // Processing 用情報として control change 80 (for user) Key (Note Number) を送信
  }

  // ピッチ角による処理分岐
  if (pitch <= -80 || pitch > 80)                // ピッチ角が±80° 以上傾いているときは、ALL NOTE OFF で全ての音を切る
  {
    if (lastEulerPitch > -80 && lastEulerPitch <= 80) // すでに送信したかどうか（複数回送信を抑制）
    {
      txMIDI_allNoteOff(0);                      // All Note Off を送信
    }
  }
  else {
    // Note ON するかどうかの判定
    if (lastEulerPitch < 0 && pitch >= 0)    // Pitch がマイナスからプラスに変化（振り下ろされた）したとき
    {
      txMIDI_noteOn(0, note, vel);           // ノート ON を送信
    }
  }
  lastNote = note;                         // 直前のノートナンバーを更新
  lastEulerPitch = pitch;                  // 直前ピッチ角を更新
}
```

図 7. 音源側 Arduino のプログラム

```
uint8_t msg[3];         // MIDI メッセージ・バッファ
uint8_t count = 0;      // MIDI メッセージ・データ数カウンター

void loop() {

  uint8_t d;            // 受信データ格納用

  while ( Serial.available() ) {     // シリアル受信データがあるとき
    d = Serial.read();               // 受信データを読み込み
    if ((d & 0xF0) >= 0x80)          // 受信データが MIDI ステータス・メッセージの場合
    {
      msg[count = 0] = d;            // メッセージ・バッファの先頭にステータスを格納
    }
    else                             // 受信データが MIDI ステータス・メッセージ以外（データ）の場合
    {
      msg[++count] = d;                      // 格納位置を更新し受信メッセージをバッファに格納

      if ((msg[0] & 0xF0) == 0xC0)           // 現在ステータスがプログラム・チェンジの場合
      {
        txMIDI_program_change(msg[0], msg[1]); // MIDI 音源にプログラム・チェンジを送信
      }
      else                                   // 現在ステータスがプログラム/チェンジ以外の場合
      {
        if (count == 2)                      // MIDI メッセージ・データを 2 バイト受信した場合
        {
          txMIDI(msg[0], msg[1], msg[2]);    // MIDI 音源に MIDI メッセージを送信
        }
      }
    }
  }
}
```

５．2 台の Arduino モジュール間での直接通信

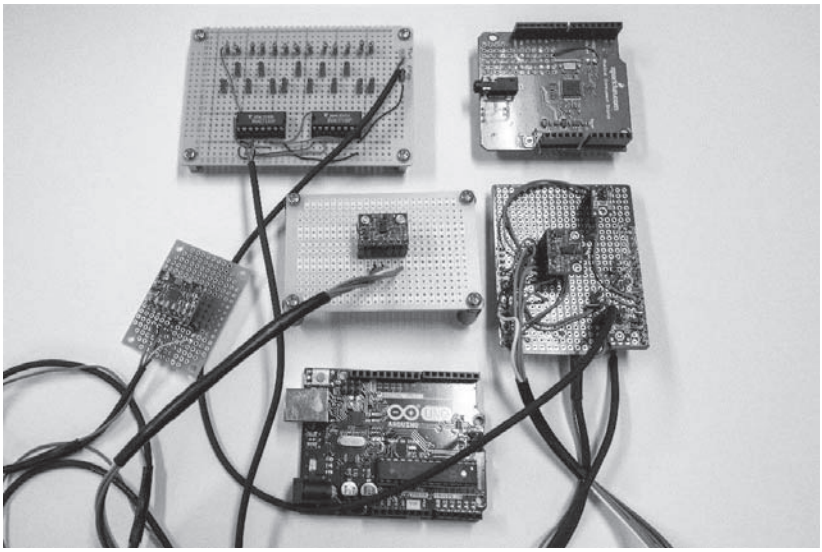
パソコンを介さず 2 台の Arduino モジュール間で直接通信を行う場合はシリアル通信ではなく，SPI や I2C による同期通信，BLE による無線通信なども考えられるが，電気的特性上の規格や通信速度の制約などもあり，慎重に検討した上でプロトタイプ機の製作に進んでいきたい。

Ⅲ．最後に

今回は Arduino によるシンプルな MIDI コントローラ製作のレポートとしたが，プロトタイプ機においては，実用性の高いコントローラの製作を目指していきたいと考えている。そのために現在，図 8

の写真にあるような複数の距離センサーやフォトリフレクタ，より小型のモーションセンサー，音階表示用の LED パネルをはじめ，様々な入出力デバイスについて検証もしており，今後これらを組み合わせ，よりよい作品製作に結びつけていきたいと考えている。

図 8. 様々な入出力デバイスの検証



注１　本稿はプロトタイプ機（評価のための一定水準の操作性を満たすもの）に向けての実験レベル段階のものであるため、現時点でビデオ映像などの公開は行わない。プロトタイプの製作が完了すれば、改めて別稿と併せ公開したいと考えている。また回路構成やプログラムも実験段階では頻繁に変更しているため、本稿で掲載しているものは、ある時点でのスナップショットとして理解いただきたい。

【参考文献】

Arduino 公式ホームページ（Arduino UNO R3 製品詳細および Arduino IDE，各種ドキュメント）
https://www.arduino.cc/ （最終確認日 2022/09/01）

Arduino 9 軸モーションシールド（回路図およびライブラリ，データシート，プルプログラム）
https://docs.arduino.cc/hardware/9-axis-motion-shield （最終確認日 2022/09/01）

SparkFun Musical Instrument shield（回路図およびライブラリ，データシート，サンプルプログラム）
https://www.sparkfun.com/products/10587 （最終確認日 2022/09/01）

Fritzing （回路図エディタ）公式ホームページ（各種ドキュメント）
https://fritzing.org/ （最終確認日 2022/09/01）

Processing 公式ホームページ（各種ドキュメント）
https://processing.org/ （最終確認日 2022/09/01）