Raspberry Pi Pico 講習会

2023 年度生用

梶本研究室

ver.1 2022.12

Change Log

1　はじめに

Raspberry Pi Pico を活用していく。執筆開始時点で WiFi 版（Raspberry Pi Pico W）は日本で発売されていないため WiFi 無し版で記載しているが途中で変更されるかもしれない。

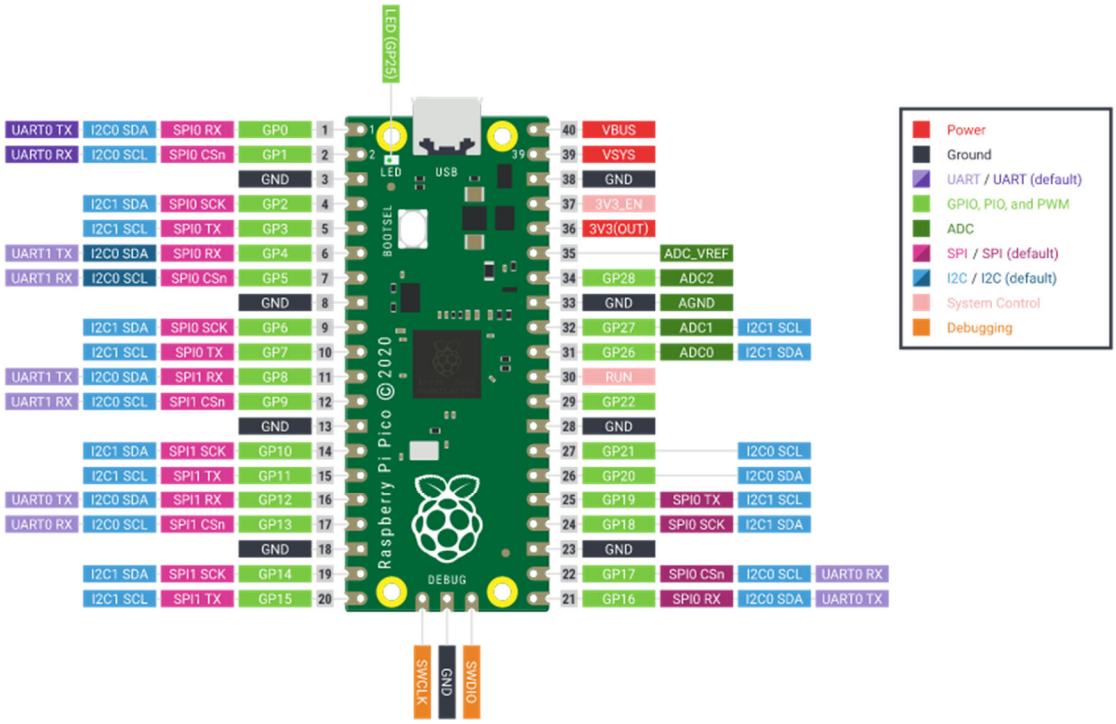


図 1　Raspberry Pi Pico ピン配置

https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

1.1 課題の進め方

- 2 章「LED の点滅」: LED を点滅させる。
- 3 章「外付けスイッチ」: 外付けスイッチによる入力を行う。割込みの練習も。
- 4 章「アナログ入力」: 加速度センサを例にアナログ入力を使う。

- 5 章「データロギング」: Processing を用い、実用的なデータロギングを行う。マルチコアについても試す。
- 6 章「RC サーボモータの駆動」: RC（ラジコン）サーボモータを動かす。
- 7 章「アナログ出力」: 内蔵の DA 変換器を使う。
- 8 章「SPI 通信」: SPI 通信によるポートの拡張例として DA 変換ポートの増設を行う。
- 9 章「DC モータ制御」: H ブリッジ IC とモータ付属のエンコーダで DC モータを PD 制御する。
- 10 章「ステッピングモータ制御」: H ブリッジ IC でバイポーラステッピングモータを動かす。
- 11 章「無線通信」: Wi-Fi のサンプルを動かす。

一週目：一日目 2 章、二日目 3 章、三日目 4 章、四日目 5 章、五日目予備日
二週目：一日目 6 章、二日目 7 章、三日目 8 章、四日目 9 章、五日目予備日
三週目：一日目 10 章、二日目 11 章
というペースで進める。

本資料中で「○○については自習すること」と書いているところは研究活動で頻出するトピックなので必ず自習してください。

1.2 レポート

最後にレポートを書く。添削します。

- Word を標準とします。Word 講習を兼ねていますので、図表番号への参照などの**参照機能**は必ず使うこと。 Word 使い方講習は講習会の途中に行う。
- 実験中に取得した図（特にオシロスコープの画面キャプチャ）を貼る。
- **回路図など、テキストにある内容を繰り返す必要はない**。
- **レポートを書きながら実験を行う**こと。これは研究の場面でも同じ。データはリアルタイムに文書化。常に写真、動画を撮るようにする。「論文を書くときに初めて写真を撮る」のは普段の研究態度として誤り。

2　LED の点滅

開発環境を整え、外付け LED を駆動する。
Pico の足がはんだ付けされていなければする（固定のためにブレッドボードを使うと良い）。
Arduino IDE を用い、Pico のライブラリを導入する。現在 2 バージョン存在しているがこの資料では Philhower 版を用いる（この場合 C++SDK ベースで作成されておりライブラリを使用できる）。例えば下記のページを参照して環境を構築すること。
https://logikara.blog/raspi-pico-arduinoide/
なおこの版の Arduino 環境に関する最もオフィシャルなドキュメントは下記になる。
https://arduino-pico.readthedocs.io/en/latest/

その後 Pico の基板に搭載された LED を点滅させる。このサンプルコードはファイル→スケッチ例→02.Digital→BlinkWithoutDelay をそのまま使えるので実際に書き込み、動作を確認する。図 2 のソースでもよい。

```
const int ledPin =  LED_BUILTIN;// the number of the LED pin

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (millis()%2000<1000) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

図 2　オンボード LED の点滅

次に LED を外付けする。図 3 を見ながら配線する。**ブレッドボードを真似るのではなく回路図から読み解くこと**。便宜上 Pico の GND（グランド）端子に複数の名前が割り当てられている（GND1,GND2,…）がこれらは内部で接続されている。

- LED の極性はテスタで確認すること。
- LED に直列に接続する抵抗（制限抵抗）はここでは 1kΩ としているが、異なる電圧の場合のためにも計算できる必要がある。抵抗値の計算方法は下記リンクなどに記載されているので自習する。照明用でなければ大体 1～10mA 流す。
  http://www.ops.dti.ne.jp/~ishijima/sei/letselec/letselec11.htm
- 抵抗のカラーコードは読めるとよい。

- ここではブレッドボードを使用。外す際には USB コネクタ等に力が加わると破損するのでマイナスドライバ等を用いる。

[課題1] LED を流れる電流を計測する（抵抗間の電圧を計測し、オームの法則により算出する）。その値は前述の計算方法で求めた予想と比較して妥当か。

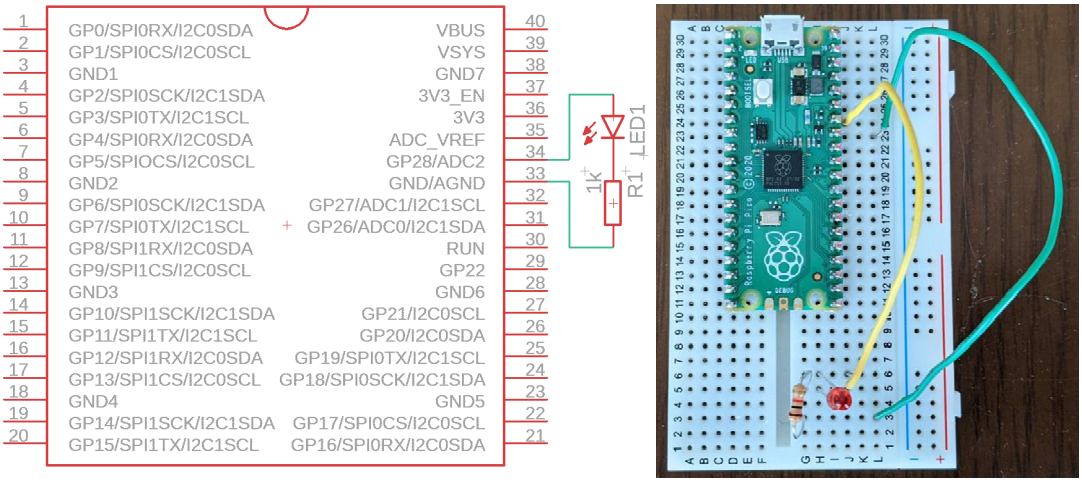


図 3　外付け LED 点滅回路

3　外付けスイッチ

外部からスイッチ入力を受け付けるようにする。図 4 参照。タクトスイッチは押下時どこが導通するかテスタでチェックしてから使用する。

回路図は簡略化のためにネット名が表記され、同じネット名は接続されていることを表す。例えば図 4 の「GND」同士、「+3V3」同士は接続される。電源と GND はブレッドボードの電源線（赤と青の線）を利用して配線する。写真のブレッドボードは電源線（右端の赤線）が二つに分かれていることに注意。

[課題2]　スイッチ回路に抵抗が必要な理由を考察する。ボタンを押したときに LOW になる回路はどのようにすればよいか。

[課題3]　タクトスイッチを押すと LED が光るようにする(digitalRead 関数)

[課題4]　割込みプログラミングについて自習し、割込みを用いたプログラムに改変する。スイッチを押すたびに LED が点滅するようにする（attachInterrupt 関数）

参考：http://gammon.com.au/interrupts

[課題5]　スイッチが押されるたびに、押された回数をシリアル通信で PC に伝えるようにする。シリアル通信の方法は自習する。PC 側は Arduino IDE のモニタ機能を用いるが RealTerm, TeraTerm などの通信ソフトを用いてもよい（文字列ではなくバイナリデータを確認する場合などに必須となる）。

RealTerm http://realterm.sourceforge.net/

TeraTerm http://sourceforge.jp/projects/ttssh2/

この際、スイッチの「チャタリング」によって押下回数が予想通りにならないと思われる。「チャタリング防止」でしらべ、コンデンサによる方法またはソフトウエアによる方法を試してみる。

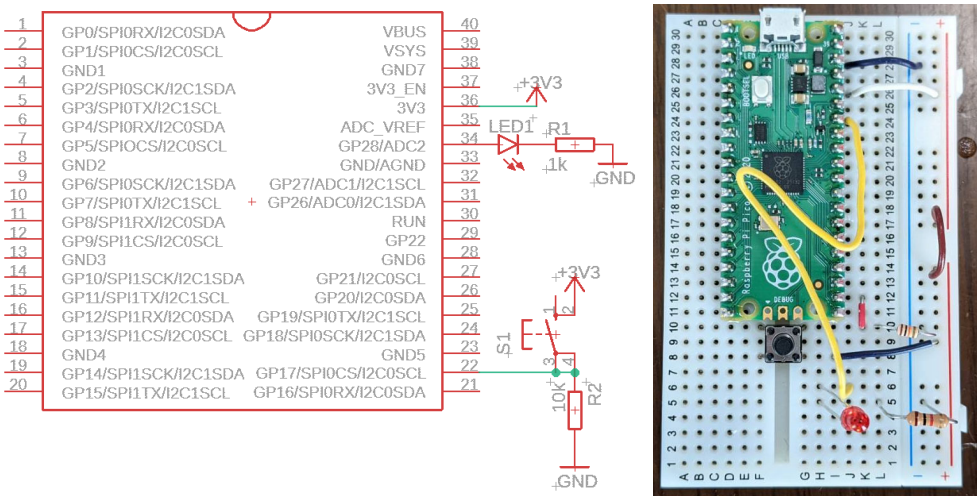


図 4　タクトスイッチ回路

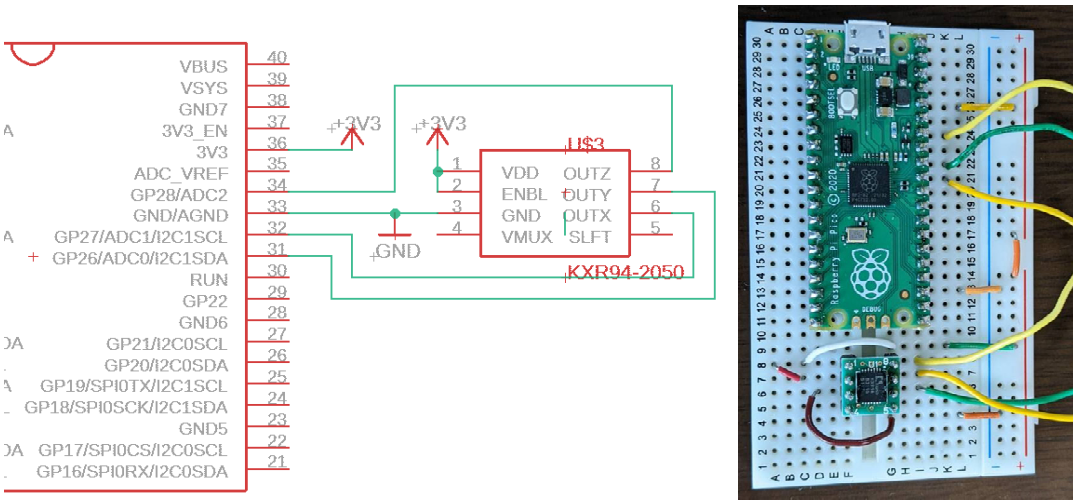
4　加速度センサ

KXR94-2050 モジュール（http://akizukidenshi.com/catalog/g/gM-05153/ ）を用いる。このモジュールは 3 軸加速度をアナログ出力するので、3 つの AD ポートで受け取れば良い。関数は analogRead。関数の引数は GPIO の番号となる。例えば AD0 は 0 番ではなく 26 番となる（これは後述する SPI 等でも同様）。

この加速度センサは 3.3V 電源に接続した場合、オフセット 1.65V、1G 加わった際に 0.66V の変化がある。つまり重力加速度の範囲では大体 0.99V～2.31V 出力が変化することになる。 Pico の AD 変換は 0～3.3V を 0～4095(12bit)に割り当てて出力する。**なおここで使用している Arduino 環境では 10bit 出力がデフォルトなので analogReadResolution 関数で 12bit に設定する**。下記のドキュメントを参照。

https://arduino-pico.readthedocs.io/en/latest/analog.html

[課題6]　PC で読み取った値が妥当であることを確認する。重力加速度が存在するため、ある軸のセンサ値は、ある向きで+1G、逆向きで-1G となるはずである。実際にどのような値で変化しているか。それは妥当か。



```
const int ADC0 = 26; //GP26
const int ADC1 = 27; //GP27
const int ADC2 = 28; //GP28

int ax, ay, az;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12); //Set the ADC resolution to 12bit
}

void loop (){
  ax = analogRead(ADC1);
  ay = analogRead(ADC0);
  az = analogRead(ADC2);
  Serial.print(ax, DEC); Serial.print(" ");
  Serial.print(ay, DEC); Serial.print(" ");
  Serial.println(az, DEC);
  delay(500);
}
```

図 5　加速度センサ回路とサンプルプログラム

5　PC プログラムによるデータロギング・Processing の導入

本章では PC プログラムを用いたデータロギングの方法を考える。
ここからは PC 側のプログラミング環境として Processing を用いる（https://processing.org/）。研究室の実験環境でも多用するため、使用したことがなければ自習したうえで以下の課題に進むこと。

[課題7]　図 6 のサンプルプログラムを写経して Pico に接続された加速度センサの電圧をシリアル通信にて PC で表示する。
その後、Processing のプログラムを変更し、データを CSV ファイル形式で保存する。PrintWriter クラスを使用する。Processing のヘルプを参照。キーボード関係の関数も調べ、どれかキーを押したら記録を開始し、100 回記録したら保存して終了するようにする。

```
const int ADC0 = 26; //GP26
const int ADC1 = 27; //GP27
const int ADC2 = 28; //GP28

int ax, ay, az;

void setup() {
  Serial.begin(115200);
  //Set the ADC resolution to 12bit
  analogReadResolution(12);
}

void loop (){
  ax = analogRead(ADC1);
  ay = analogRead(ADC0);
  az = analogRead(ADC2);
  Serial.write(ax>>8);
  Serial.write(ax&0xFF);
  Serial.write(ay>>8);
  Serial.write(ay&0xFF);
  Serial.write(az>>8);
  Serial.write(az&0xFF);
  delay(20);  //wait for 20ms
}
```

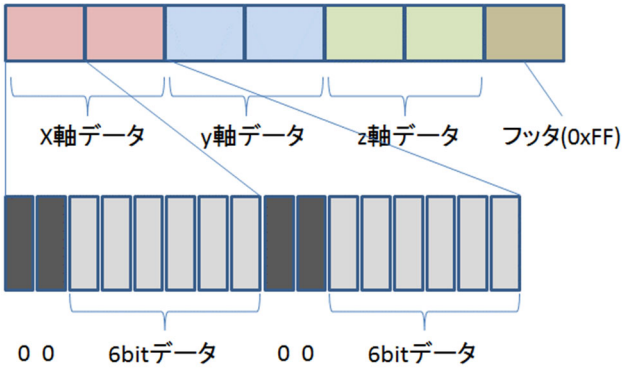
```
import processing.serial.*;

Serial myPort;
String COM_PORT="COM9"; //COM番号. 変更する
int time=0;
int ax=0,ay=0,az=0;

final int WINSIZE=512;

void settings(){
  size(WINSIZE, WINSIZE);
}

void setup() {
  frameRate(60);
  //シリアルポートに接続. 速度は115.2kbpsに設定
  myPort = new Serial(this, COM_PORT, 115200);
}

void draw() {
  color cx = color(255,0,0);
  color cy = color(0, 255,0);
  color cz = color(0, 0, 255);

  //シリアル通信
  if(myPort.available()>=6){
    ax = myPort.read() * 256;
    ax = ax + myPort.read();
    ay = myPort.read() * 256;
    ay = ay + myPort.read();
    az = myPort.read() * 256;
    az = az + myPort.read();
  }
  //描画
  time = (time+1)%WINSIZE;
  fill(cx);ellipse(time,ax/16, 5, 5);//x座標
  fill(cy);ellipse(time,ay/16, 5, 5);//y座標
  fill(cz);ellipse(time,az/16, 5, 5);//z座標
  println(ax," ", ay," ",az);
}
```

図 6　加速度センササンプルプログラム（左）Arduino 側、(右)Processing 側

上記サンプルでは Processing 中の Draw 関数中でシリアル通信を行った。しかし Draw 関数は通常 60fps で読み出される関数である。より高頻度ないし正確なデータ記録をする場合には限界がある。

Processing のシリアルライブラリはこうした場合に有用な仕組みをもっている。まず serialEvent 関数を定義する。また bufferUntil 関数を用いて、特定の文字が来た時にだけ serialEvent が発生するようにする。これらは関数のヘルプで使い方を調べること。この仕組を使うためには、データの末尾に特定の文字（以後フッタと呼ぶ）を指定する必要があるため、一回あたりの送受信量は 1byte 増える（図 7）。

またこの時、フッタを示す数値がデータ中に偶然現れることを避ける必要がある。今回の AD 変換は 1ch あたり 12bit であるので、この上位、下位 6bit ずつ送信すれば、データはすべて 0-63 の範囲に収まる。フッタの数値を例えば 0xFF(8bit すべて 1)とすれば、データと指定文字が重なる心配はなくなる。




```
const int ADC0 = 26; //GP26
const int ADC1 = 27; //GP27
const int ADC2 = 28; //GP28

int ax, ay, az;

void setup() {
  Serial.begin(115200);
  //Set the ADC resolution to 12bit
  analogReadResolution(12);
}

void loop (){
  ax = analogRead(ADC1);
  ay = analogRead(ADC0);
  az = analogRead(ADC2);
  Serial.write(ax>>6);
  Serial.write(ax&0x3F);
  Serial.write(ay>>6);
  Serial.write(ay&0x3F);
  Serial.write(az>>6);
  Serial.write(az&0x3F);
  Serial.write(0xFF); //footer
  delay(5);  //wait for 5ms
}
```

図 7　（左）データ列とフッタの例（右）Arduino コード

複数バイトのデータを読み出すに際して、データの「切れ目」が正しいことを保証する事が重要である。マイコンからのデータは、マイコンを起動するタイミング、PC のプログラムを起動するタイミング、などにより、Y 軸データから始まっているかもしれないし、Z 軸データの２バイト目から始まっている可能性もある。

この問題を解決するためには、第一に PC 側からデータ送信のリクエスト信号をマイコン側に送る方法が考えられる。これが最も標準的だが、図 8 のように PC 側のプログラムでフ

ッタを明示的に探索する方法もある。これによって初回の読み出し時にデータの切れ目が正しく認識される。

```
void setup() {  // setup() runs once
  int i;
  size(256, 256);
  frameRate(60);
  //シリアルポートに接続. 速度は115.2kbpsに設定
  myPort = new Serial(this, COM_PORT, 115200);
  myPort.bufferUntil(0xFF);
  output = createWriter("data"+year()+month()+day()+hour()+minute()+".csv");
  noStroke();
}
```

```
void serialEvent(Serial p) {

  ax = p.read() * 64; //upper 6bit
  ax = ax + p.read();
  ay = p.read() * 64;
  ay = ay + p.read();
  az = p.read() * 64;
  az = az + p.read();
  while(p.read() != 0xFF);//フッタをチェック.

  if(WriteToFile==true){
    output.println(ax+","+ay+","+az);
  }
}
```

図 8　Processing 内のシリアルイベントの設定（一部）

bufferUntil による割り込み関数の定義は、こうした割り込み関数の常として、serialEvent 関数がいつ呼び出されるかわからないために再現性のないバグを生むことがある。例えば serialEvent 関数は、bufferUntil が呼ばれた次の瞬間に（まだ Processing の setup 関数が終わらないうちに）呼ばれるかもしれない。例えばこれによって、まだ書き込むファイルが準備されていない状態でファイル書き込みが生じる等のバグが生じる。こうしたバグは典型的に「何回かに一回」発生する。こうしたバグに対してアドホックな解決は取らないこと。

[課題8]　加速度センサの 3 軸の値を 1kHz で記録したい。上記の解説を参考に、Pico 側から 1kHz（1ms 周期）で AD 変換、シリアル通信を行い、データを記録せよ。その後 Pico 側のデータ取得間隔を正確にするため図 9 を参考にタイマー割り込みを行う。**1ms 周期で計測できていることを確認するため、適当なデジタル出力端子の状態を毎回反転させ、オシロスコープで観察する。**（図 9 では GP16 を観察用ピンとしている。またシリアル通信速度を高めに設定している）

（以降**オシロスコープで観察と書いてある場合はその画面をオシロの機能でキャプチャし、レポートに添付すること。**オシロスコープの使い方は次の資料を参照：
http://download.tek.com/document/55Z-17291-3.pdf
http://download.tek.com/document/3GZ-24924-0.pdf）

繰り返しタイマー割込みに関しては C++ SDK Documantation 等を参照。

https://raspberrypi.github.io/pico-sdk-doxygen/group__repeating__timer.html

http://igarage.cocolog-nifty.com/blog/2022/04/post-0517b2.html

```
const int ADC0 = 26; //GP26
const int ADC1 = 27; //GP27
const int ADC2 = 28; //GP28
const int CheckPin =  16;// GP16

struct repeating_timer st_tm1ms;
int ax, ay, az;
int chk = 0;

bool tm1ms(struct repeating_timer *t)
{
  chk = !chk; //For oscilloscope observation
  digitalWrite(CheckPin, chk);
  ax = analogRead(ADC1);
  ay = analogRead(ADC0);
  az = analogRead(ADC2);
  Serial.write(ax>>6);
  Serial.write(ax&0x3F);
  Serial.write(ay>>6);
  Serial.write(ay&0x3F);
  Serial.write(az>>6);
  Serial.write(az&0x3F);
  Serial.write(0xFF); //footer
  return true;
}

void setup() {
  pinMode(CheckPin, OUTPUT);
  Serial.begin(921600);
  //Set the ADC resolution to 12bit
  analogReadResolution(12);
  //Repeating timer interrupt at 1ms
  add_repeating_timer_us(1000, tm1ms, NULL, &st_tm1ms);
}

void loop (){
  //do nothing
}
```

図 9　タイマー割り込みによる 1ms 計測のサンプルプログラム

上記は Processing 側の Draw 関数の周期に依存しない計測の方法である。ただし PC を入れた制御ループを構成する際には USB 通信の安定性が問題となる。**高速な制御ループは可**

能ならマイコン内で完結させるべきである。
振動子の応答特性計測などではより速い計測が必要となるが、リアルタイム性が必要でない場合には、マイコン上のメモリに一時保存することもよく行われる。

5.1 マルチコア（参考）

行うタスクがより多い場合やデータの送信量が多い場合、通信によるオーバーヘッドが問題となる（数百点のセンサデータを用いる場合など）。

そうした場合、Pico が CPU コアを２つ持っていることを利用し、「シリアル通信用のコア」と「計測・制御用のコア」に分けることで安定化させることが出来る。例えば計測・制御は 1kHz で行い、シリアル通信は PC からの指令値を受け取ったり、PC に現在の状態を送信したりする（こちらも 1kHz で良いが、目的によってはより低い周期でも構わない）。

図 10 はマルチコアのサンプルプログラムの追加部分である。setup1()関数、loop1 関数で簡単に実現できる。この例では PC からデータを受け取ると特定のデジタル出力ピンを反転している。

マルチコアは特に課題とはしないので、実課題で必要な際に参照すること。

```
void setup1(){
}

void loop1(){
  if(Serial.available()>0){
    Serial.read();
    chk = !chk; //For oscilloscope observation
    digitalWrite(CheckPin, chk);
  }
}
```

図 10　マルチコアのサンプルプログラム（追加部分）

6　RC サーボモータ駆動（PWM）

PWM 出力を利用して RC（ラジコン）サーボモータを動かす。

ここから基板にはんだ付けしていく。使用する IC は基板に直接はんだ付けせず、ソケットを使う。研究室では IC を使いまわしているので大切に。Pico の場合は 40 ピンのソケットを折って使う。RC サーボモータの駆動には USB 給電の 5V を利用するが、複数台を駆動する場合や大型の RC サーボの場合は外部電源を用いる。使用している RC サーボは RB50 (http://www.ministudio.co.jp/Cgi-bin/Order-JP/DetailJp.asp?GoodsNum=202) 。ケーブルの色と配線を確認する。

図 11 に示すように RC サーボモータを接続する。3 ピンのピンヘッダを用いる。

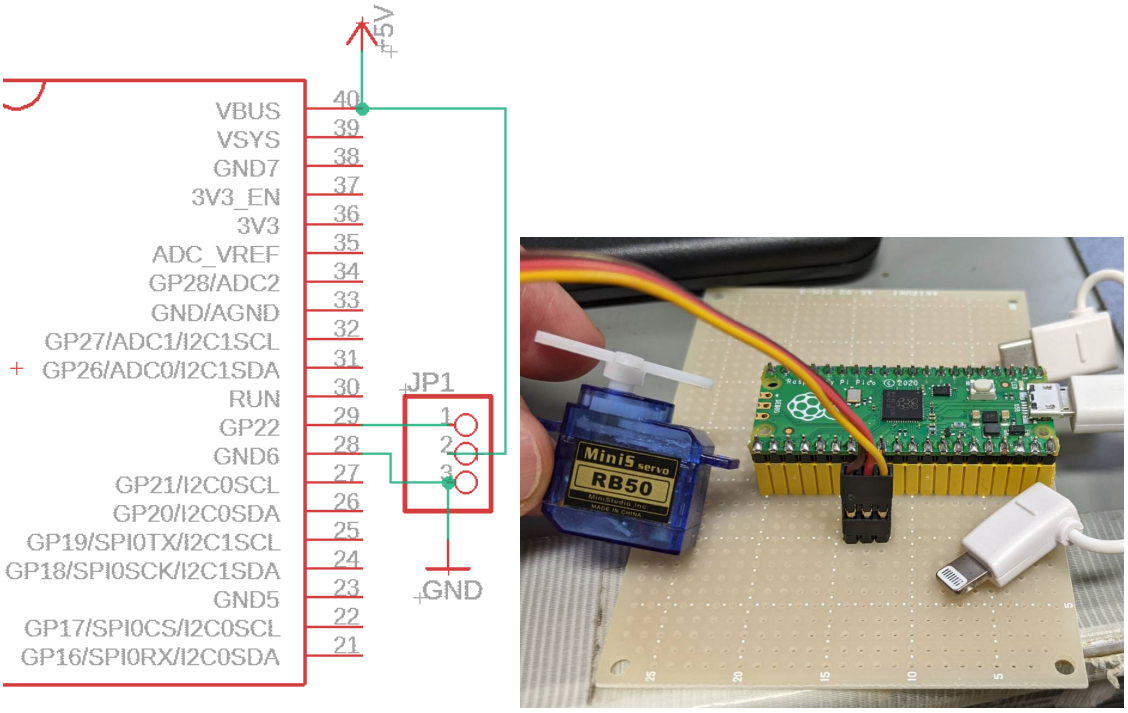


図 11 RC サーボ回路

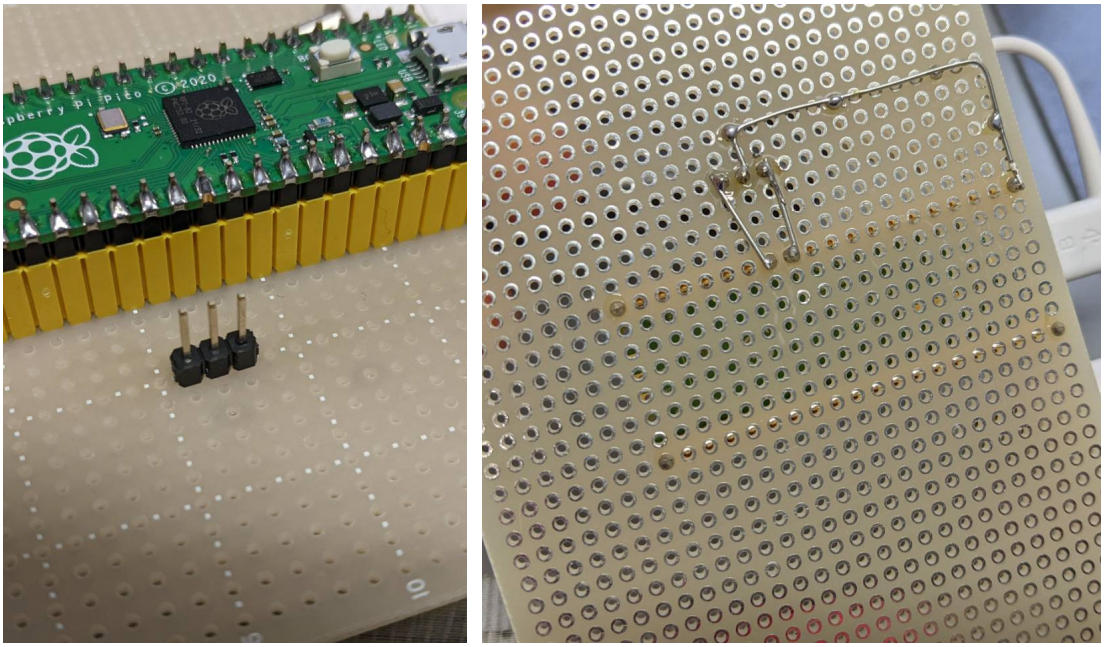


図 12 ピンヘッダ、ソケットのはんだ付け

- 配線は錫メッキ線、あるいはジュンフロン線を用いる。無用に太い線は使わない。ただしモータを駆動する部分や電源部分には太い線を用いる。
  - ハンダ付けの方法は下記リンクや「電子工作の素」などを参照する。
  - ハンダ付けの方法　http://www.youtube.com/watch?v=S5f7jueQHr8
  - 良いハンダ付け形状など　https://godhanda.co.jp/blog/kisokouza08/

PWM および RC サーボモータの制御方法は例えば下記で自習。

- http://berry.sakura.ne.jp/technics/servo_control_p1.html
- http://startelc.com/H8/H8_17RCserv.html

RC サーボライブラリを使う。
https://docs.arduino.cc/learn/electronics/servo-motors
https://www.arduino.cc/reference/en/libraries/servo/
こちらのページの”SWEEP”というサンプルを使えば動かすことができる。ピン番号は適切に変更する。

[課題9]　PC からのシリアル通信でサーボモータの姿勢を制御する。例えば 0-9 の数値を送る。PC 側はシリアル通信ソフトで良い。オシロスコープで PWM 信号を観察する。パルス幅、パルス間隔は計算どおりになっているか

シリアル通信で PC からの入力を待つためには available 関数を用いる。典型的には図 13 のようになる。

```
void loop() {
  int incomingByte;

  if(Serial.available()>0){
    incomingByte = Serial.read();
    ここで送られてきたキーに応じた処理
  }
  delay(15);
}
```

図 13 RC サーボとシリアル通信

7　アナログ出力（PWM＋ローパスフィルタ）

Pico は通常のアナログ出力を持っておらず、代わりに PWM（パルス幅変調）によって疑似的なアナログ出力を可能としている(analogWrite 関数)。LED の調光等ではこれは問題にならないが、アナログ電圧出力を必要とする場合には困る。

ここでは簡易的に、PWM 出力に抵抗とコンデンサによるローパスフィルタ（LPF）をかけることでアナログ出力を得る。Pico の場合、PWM 周波数は analogWriteFreq 関数で変更できるため、これを 60kHz まで上げて音声領域に対応させる。また analogWrite 関数はデフォルトでは 0 から 255 までの数値を受け付けるが、analogWriteRange 関数によってビット数を変更することができる（今回は使わない）。

https://arduino-pico.readthedocs.io/en/latest/analog.html

[課題10]　PWM について自習する。その後、図 14 のように analogWrite 関数を使って DA 出力を行い、出力波形をオシロスコープで確認する。

このプログラムは PWM によって 100Hz の正弦波を表現しようとしており、**正確な周波数を実現するために micros 関数を使っている。** micros 関数はプログラム起動時から現在までの時間をマイクロ秒単位で返す関数であり、これを 1/1000000 すれば現在の時刻が秒単位で得られる。例えば f[Hz]の周波数の音を出したい場合、現在の時刻が t[s]であれば、sin(2πft)を出力すれば良い。

この例では PWM 波形は GP21 から出力されるので、GP21 をオシロスコープで計測すればよい。どのような波形となり、その周期は PWM の設定から妥当か確認する。

```
const int analogOutPin = 21; //GP21
unsigned long t;
int outputValue = 128;        // value output to the PWM

void setup() {
  analogWriteFreq(60000); //PWM set to 60kHz
}

void loop() {
  t = micros();
  outputValue = (int)(128.0 * sin(2.0 * PI * 100.0 * (float)t/1000000.0)+127.0);
  analogWrite(analogOutPin, outputValue);
}
```

図 14　analogWrite 関数による PWM 出力のサンプルプログラム

次にオペアンプを用いたローパスフィルタ回路を使い、PWM 出力をなだらかなアナログ出力に変換する（図 15）。この回路では次のことが行われている。

① オペアンプ（新日本無線 NJM4580）の反転増幅回路（兼ローパスフィルタ）で増幅。増幅率は R1、R2 で指定。

② R2 と C1 によるローパスフィルタ。ここではカットオフ周波数は fc=1/(2πRC)=1.6kHz としている。
③ 増幅時のオフセット電圧を R3 と R4 で設定し(2.4V)、オペアンプの V+に入力。これにより単電源での増幅を可能としている。
④ C3 とイヤフォンの抵抗によるハイパスフィルタで出力の直流成分カット、イヤフォンジャックからイヤフォンやスピーカに出力。

- 電解コンデンサの極性に注意。＋側をオペアンプの出力側とする（これはなぜか）
- イヤフォンジャックは基板に直付けできるタイプを用いる。ステレオジャックの場合接続に注意。https://akizukidenshi.com/download/ds/switronic/ST-005-G.pdf
- コンデンサの数値コードは読めるようにする。
  http://www.jarl.org/Japanese/7_Technical/lib1/konden.htm

オペアンプ回路については下記などで自習する。 少なくともバーチャルショート（仮想接地）を使った計算ができること。
https://kaji-lab.jp/ja/index.php?plugin=attach&pcmd=open&file=OpAmpIntro.pdf&refer=people%2Fkaji
https://www.marutsu.co.jp/pc/static/large_order/opamp_20220315
https://www.marutsu.co.jp/pc/static/large_order/opamp2_20220329
https://engineer-education.com/operational-amplifier_application-circuit/
https://www.fbnews.jp/202111/junkshop/
https://www.analog.com/jp/education/landing-pages/003/opamp-application-handbook.html (バイブルだがこの講習には too much)

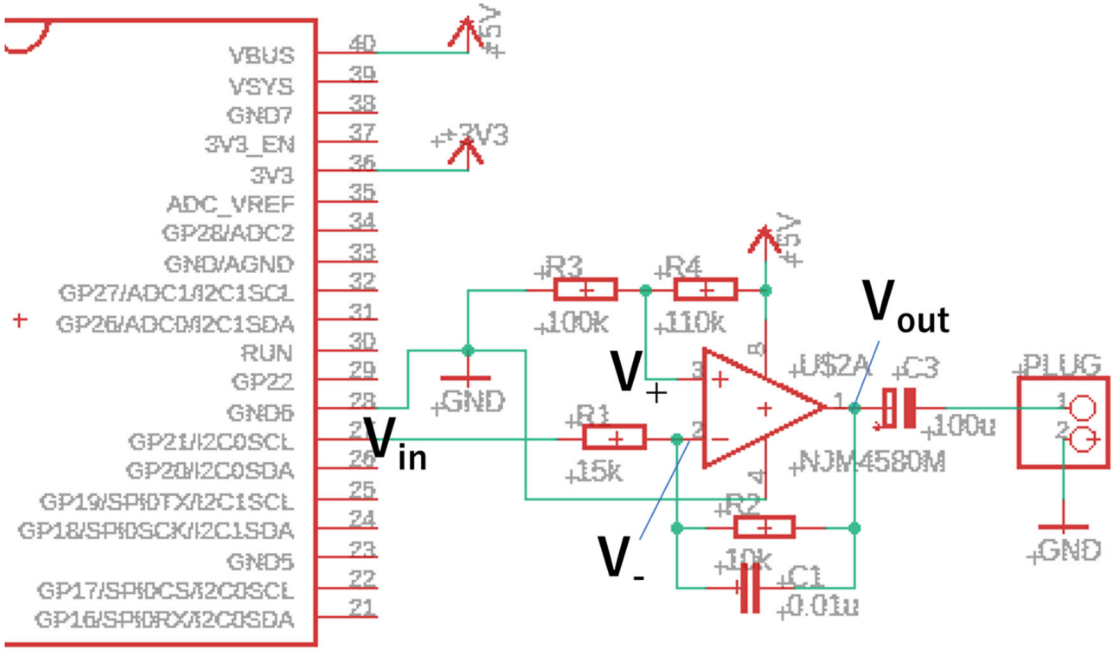


図 15　アナログ出力とオペアンプ回路

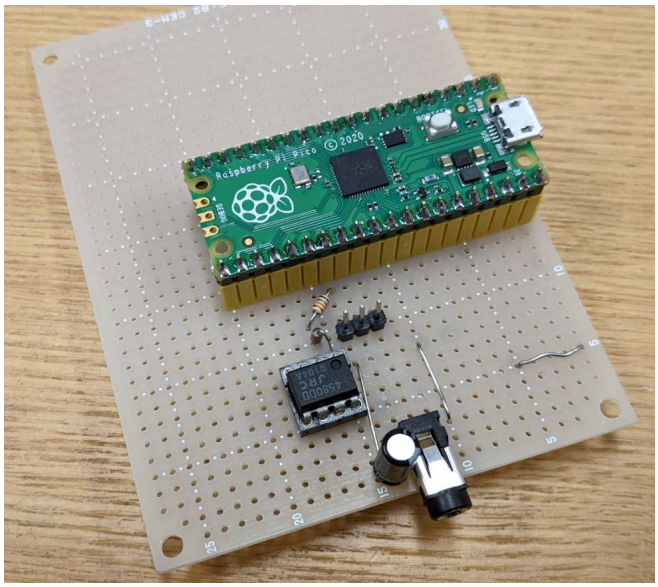


図 16　外観

オペアンプを用いたローパスフィルタの解説は例えば下記を参照。

https://detail-infomation.com/opamp-low-pass-filter/

図 15 の回路の各定数は次のように決められている。

コンデンサを無視した場合の増幅機能、オフセット調節機能は次のように計算される。

入力をV_{in}、出力V_{out}とする。バーチャルショートの原理から、$V_+ = V_-$だから、

$$V_+ = V_- = V_{in} + \frac{R_1}{R_1 + R_2}(V_{out} - V_{in}) = \frac{R_2}{R_1 + R_2}V_{in} + \frac{R_1}{R_1 + R_2}V_{out}$$

$$(R_1 + R_2)V_+ = R_2 V_{in} + R_1 V_{out}$$

$$V_{out} = \frac{R_1 + R_2}{R_1}V_+ - \frac{R_2}{R_1}V_{in}$$

ここで、$0V \leq V_{in} \leq 3.3V$である。また<u>このオペアンプはカタログスペック上、正負電源電圧に対して 1V 程度の余裕が必要</u>なため、$1V \leq V_{out} \leq 4V$となることが望ましい（こうした余裕が不要なオペアンプもある。Rail-to-Rail で検索）。つまり 0～3.3V の入力を 1～4V の出力に変換するために R1,R2 等を決めていくことになる。この時ゲインは 1 よりやや小さくないといけないから、R1=15kΩ、R2=10kΩとする。その前提で出力が 1～4V に収まるようにするために、$V_+ = 2.4V$となるようにする（R3=100kΩと R4=110kΩで 5V を分圧）。そうすると

$$V_{out} = \frac{R_1 + R_2}{R_1}V_+ - \frac{R_2}{R_1}V_{in} = 4 - 0.67V_{in}$$

となる。この時$1.79V \leq V_{out} \leq 4V$となる。

[課題11]　図 15 に示した回路を作成し、V_{out}の場所における波形をオシロスコープで観察する。振幅、オフセット電圧はそれぞれいくつか。それらは設計（上記解説）と比べて妥当か。

[課題12]　イヤフォンを接続し、音を聞いてみる。C3 とイヤフォンの抵抗によるカットオフ周波数はいくつか。イヤフォンの抵抗値が 32Ωのとき、C3 とイヤフォンによるハイパスフィルタによるカットオフ周波数はいくつか（抵抗とコンデンサによるハイパスフィルタを自習すること）。

<u>このような典型的なオーディオ回路は、イヤフォン、スピーカなどの負荷が接続されてはじめてハイパスフィルタの特性が決定されることに注意。</u>負荷を接続せずに計測して波形が想定と異なるというミスがよくある。

なお今回はイヤフォンを鳴らすためにオペアンプを用いたローパスフィルタを作成したが、<u>アナログ波形を得ることだけが目的であれば抵抗とコンデンサのみでローパスフィルタとしても良い。</u>

[課題13]　PC からのシリアル通信でイヤフォンの音を制御する。1-8 の数値を送ると「ドレミファソラシド」が鳴るようにする。PC 側はシリアル通信用ソフトで良い。オシロスコープで波形の変化を確認する。

8　外付けの DA（SPI 通信によるポートの拡張）

Pico には SPI 通信モジュールが 2 つ用意されている。これを使うことでポートを拡張することが出来る。ここでは 8ch の 10bit D/A ポートを実現する。
SPI 通信については下記で自習。シフトレジスタを理解していることが前提なので、そちらも自習する。
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

外付けの DA 変換器として 8ch10bit D/A コンバータ LTC1660CN を用いる（図 17）。**この演習はデータシートをきちんと読む練習を兼ねているので手元にデータシートを用意。**
DA 変換器の電源と GND の間に電圧安定化のためのコンデンサを入れる。10pin 端子を使い、余った 2pin は GND と 5V にしている。
http://akizukidenshi.com/catalog/g/gI-02794/

通常の Arduino の SPI 通信と異なる点として、Pico の場合はピンを変更する関数が用意されている（SPI.setRX、 SPI.setCS、SPI.setSCK、SPI.setTX）。またチップセレクト（CS）信号をハード的にサポートするためには SPI.begin 関数の引数を true とする必要がある（ハード的にサポートしない場合はスケッチ中で明示的に CS ピンを変化させる必要がある。以上の情報は下記に記載されている。
https://arduino-pico.readthedocs.io/en/latest/spi.html

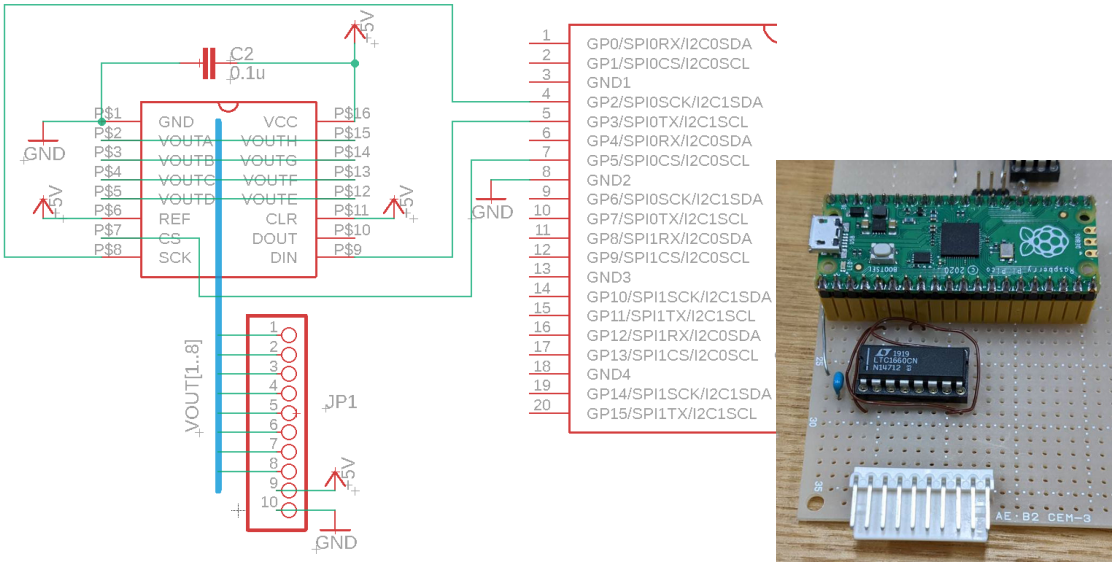


図 17　8ch DA ポート回路（10pin の端子を使い余った 2pin は GND と 5V に）

[課題14]　　図 18 のサンプルプログラムを動かし、オシロスコープを使って波形を観察す

る。つぎにクロック信号とデータ信号を観察する（３つのプローブを使い、CS ピンをトリガ信号とする。これはオシロスコープにおけるトリガの使い方の練習を兼ねる）。

http://download.tek.com/document/55Z-17291-3.pdf

http://download.tek.com/document/3GZ-24924-0.pdf

二つの信号のタイミングはどのような関係にあるか。それらは LTC1660 のデータシートのタイミングチャートと比較して妥当か。

16bit のデータ信号はプログラムとどのような関係にあるか。またそれらは LTC1660 のデータシートと比較して妥当か。

```
#include <SPI.h>
const int SPI_SCLK=2; //GP2
const int SPI_MOSI=3; //GP3
const int SPI_MISO=4; //GP4, not used
const int SPI_CS=5;   //GP5

short DA,spiData;
char channel = 0;
int t=0;
unsigned long time;

void setup() {
  Serial.begin(9600);
  SPI.setRX(SPI_MISO);
  SPI.setCS(SPI_CS);
  SPI.setSCK(SPI_SCLK);
  SPI.setTX(SPI_MOSI);
  SPI.begin(true); //true = hardware CS, false = software CS
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
}

void loop() {
  t = (t+1)%2;
  if(t==0){
    DA = 0x3FF;
  }else{
    DA = 0;
  }
  spiData = (((channel&0x07)+1)<<12) | ((int)(DA&0x3FF)<<2);
  SPI.transfer16(spiData);
}
```

図 18 8ch D/A 変換器 LTC1660 を動かすサンプルプログラム

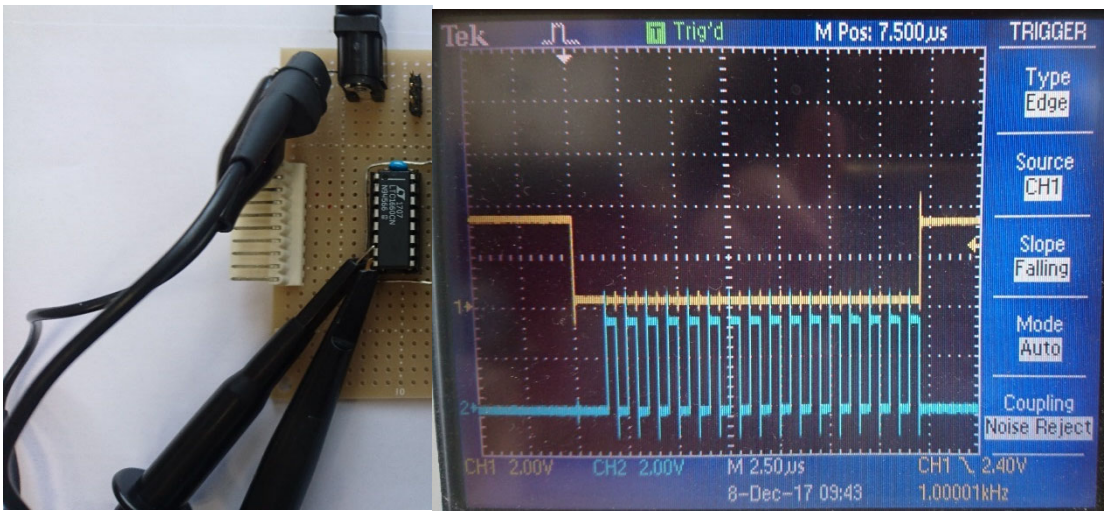


図 19 制御信号の観察

[課題15] チャンネルを指定して電圧を出力できる関数を作成する。

void DAout(char channel, float voltage)

LTC1660 のデータシートを見ながらチャンネル指定の方法を考える。電圧を 0.0 から 5.0 までの float で指定出来るようにする。その上で、8ch それぞれから指定した異なる周波数の正弦波を出力し、オシロスコープで波形を観察する。正確な周波数で出力するために[課題 13] の方法を用いる。

ただし 8ch 分の sin 関数をリアルタイムに計算するには時間がかかり loop 周期が遅くなるかもしれない。この問題を解決するためにはあらかじめ setup 関数中で正弦波のテーブル（配列）を作成しておく工夫が必要かもしれない。

9　DC モータの制御

MAXON 社製 DC モータ(RE25, 10W, 118746)を駆動する。使用する DC モータにはエンコーダ(HEDS5540、500CPR (CPR: count per revolution))が付いており、位置を計測しながら制御する事ができる。出力にはモータドライバ IC の DRV8835 を使用する。このモータドライバは 2ch 分駆動できるが、この章では 1ch のみ使用する。

https://www.maxongroup.co.jp/maxon/view/product/motor/dcmotor/re/re25/118751

https://www.maxongroup.co.jp/maxon/view/product/sensor/encoder/Optische-Encoder/ENCODERHEDS5540/110511

9.1 エンコーダの動作

まずエンコーダを動作させる。エンコーダについては例えば下記で自習する。**特に 4 逓倍モードについて理解する**。

http://ednjapan.com/edn/articles/1203/16/news012_2.html

https://www.sk-solution.co.jp/html/qa/qad001c00820090629163814.html

図 20 図 21 図 22 図 23 を参考に回路を作成し、モータのエンコーダと自作ケーブルにて接続する。この時点ではエンコーダ部分のみ作成すれば良い。5V、GND,チャンネル A,B を接続する。各出力チャンネルは 5V 電源と 3.3k オームの抵抗で接続する（これを**プルアップ抵抗**という：自習する)。実際にはプルアップ抵抗無しでも動くが高速回転の際に不具合を生じることがある。500 パルス／一周のエンコーダを 4 逓倍モードで利用している。

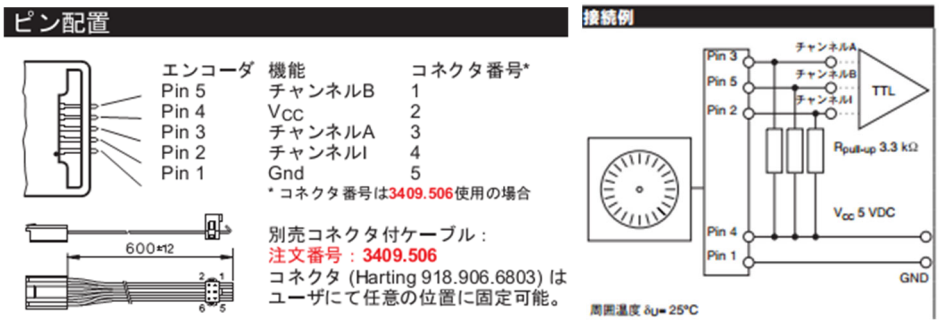


図 20 エンコーダピン配置（HEDS5540 のマニュアルより）

Arduino 環境のエンコーダライブラリについては下記に大量のリストがある。ただし 2022 年 12 月時点で Pico 用に使いやすい形でライブラリが提供されていないと思われるため、今回は下記ページの「Interrupt Example (the Encoder interrupts the processor). Uses both Interrupt pins」というセクションにかかれているサンプルプログラムを利用する。このサンプルは I/O ピンの状態変化によって割り込みを発生させている。

https://playground.arduino.cc/Main/RotaryEncoders

実際のソースコードは図 24 を参照。上記のサンプルから多少改変している（IO ピンを変

更、カウント用の変数を符号付き整数に変更)。

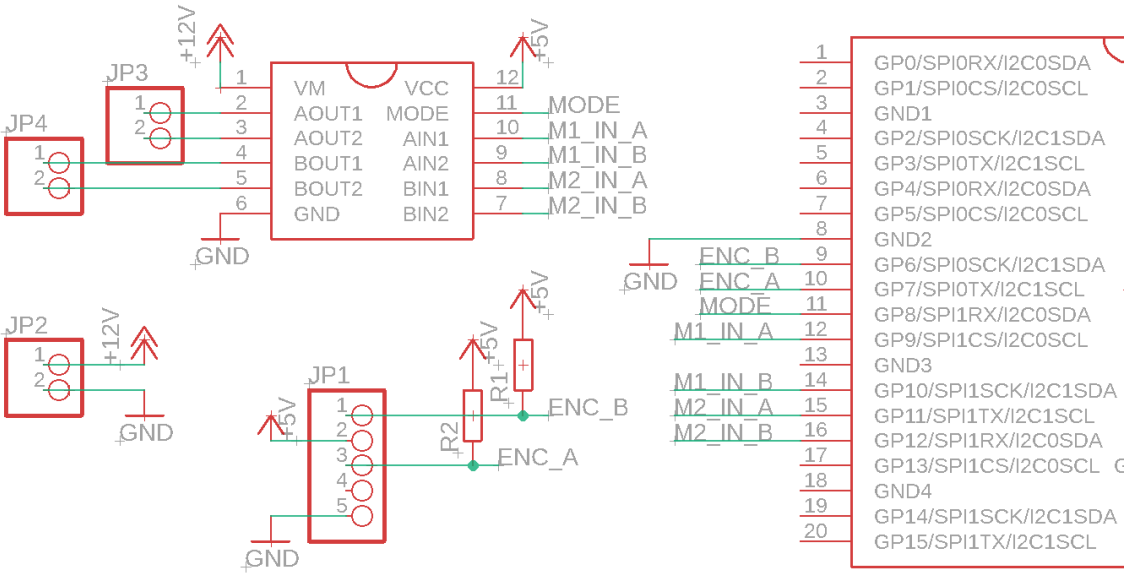


図 21　エンコーダ読み取り回路およびモータドライバ回路

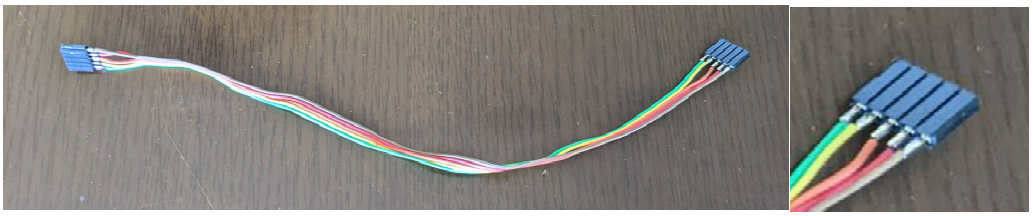


図 22　エンコーダ用ケーブル（自作）

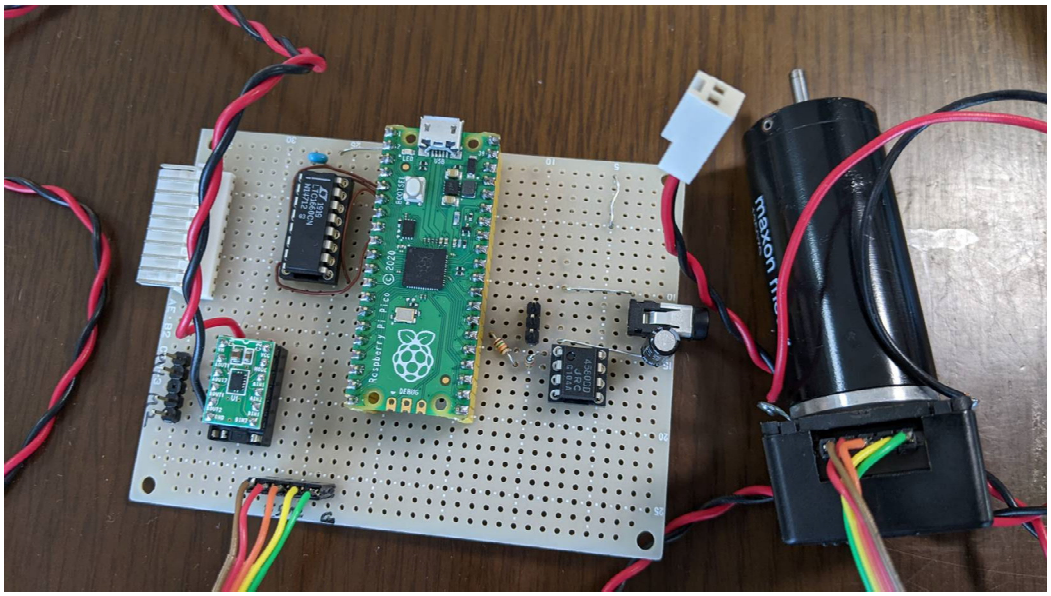


図 23　エンコーダ・モータドライバ回路

```
//Sample encoder code copied from
// https://playground.arduino.cc/Main/RotaryEncoders/
// section: Interrupt Example (the Encoder interrupts the processor). Use
#define encoder0PinA  7
#define encoder0PinB  6
volatile int encoder0Pos = 0; //正負の値を取れるよう変更

void setup() {
  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);
  attachInterrupt(encoder0PinA, doEncoderA, CHANGE); //割り込み関数指定
  attachInterrupt(encoder0PinB, doEncoderB, CHANGE); //割り込み関数指定
  Serial.begin (9600);
}
void loop() {
  //Do stuff here
  Serial.println(encoder0Pos);
  delay(500);
}
void doEncoderA() {
  // look for a low-to-high on channel A
  if (digitalRead(encoder0PinA) == HIGH){
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == LOW){
      encoder0Pos = encoder0Pos + 1;          // CW
    }else{
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }else{// must be a high-to-low edge on channel A
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == HIGH){
      encoder0Pos = encoder0Pos + 1;          // CW
    }else{
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }
  //Serial.println (encoder0Pos, DEC); //デバグ用：コメントアウト
}
void doEncoderB() {
  // look for a low-to-high on channel B
  if(digitalRead(encoder0PinB) == HIGH){
    // check channel A to see which way encoder is turning
    if(digitalRead(encoder0PinA) == HIGH){
      encoder0Pos = encoder0Pos + 1;          // CW
    }else{
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }else{// Look for a high-to-low on channel B
    // check channel B to see which way encoder is turning
    if(digitalRead(encoder0PinA) == LOW){
      encoder0Pos = encoder0Pos + 1;          // CW
    }else{
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }
}
```

図 24　エンコーダを読むサンプルプログラム

[課題16]　　エンコーダを用い、モータの回転角度を計測、シリアル通信によって PC 画面に表示し続けるようにする。角度はラジアンに変換して表示する。正転・逆転方向に一周させて±2πの数値が表示されるか確認する。

9.2 モータの駆動

次にモータを駆動する。H ブリッジ回路については例えば下記で自習する。

http://www.picfun.com/motor03.html

モータ駆動（H ブリッジ駆動）は PWM 制御により明示的に制御する。今回使用する DRV8835 は下記のものである。

https://akizukidenshi.com/catalog/g/gK-09848

https://akizukidenshi.com/download/ds/akizuki/AE-DRV8835-S_20210526.pdf

「動作モード」と「ピンの名称と機能」の項を見ること。DC モータを駆動する場合、MODE ピンを GND に接続すればよい。この場合、正転、逆転、ブレーキ、空転モードが存在する。今回は、次の章でステッピングモータの駆動時に MODE ピンを HIGH にする必要があるため、ディジタル出力ピンに接続している。

■動作モード■

IN/IN モード(MODE = 0)					
MODE	xIN1	xIN2	xOUT1	xOUT2	動作
0	0	0	HiZ	HiZ	空転
0	0	1	L	H	逆転
0	1	0	H	L	正転
0	1	1	L	L	ブレーキ
PASE/ENABLE モード(MODE = 1)					
MODE	xENABLE	xPAHSE	xOUT1	xOUT2	動作
1	0	X	L	L	ブレーキ
1	1	1	L	H	逆転
1	1	0	H	L	正転

■ピンの名称と機能■ (ピン番号は基板左上を起点に反時計回りです)

ピン	名称	機能	ピン	名称	機能
1	VM	モータ電源	12	VCC	ロジック電源
2	AOUT1	A出力1	11	MODE	モード設定
3	AOUT2	A出力2	10	AIN1	A入力1/APHASE
4	BOUT1	B出力1	9	AIN2	A入力2/AENBL
5	BOUT2	B出力2	8	BIN1	B入力1/BPHASE
6	GND	グランド	7	BIN2	B入力2/BENBL

■絶対最大定格■

図 25　　DRV8835 の動作モードとピンの役割。

https://akizukidenshi.com/download/ds/akizuki/AE-DRV8835-S_20210526.pdf

図 20 図 21 図 22 図 23 を参考に回路の残りの部分を作成する。モータ駆動用の電源は外部電源を用意する（回路図中の＋12V と書かれている部分）。GND 同士の接続を忘れないこと。

モータ駆動用電源は実験用の安定化電源を用いる。この電源は電圧設定、および電流リミットの設定が可能である（**CVCC 電源**: Constant Voltage Constant Current power supply）。一般的な働きとして、制限電流未満のときは定電圧電源として働き、それを上回ったときは定電流電源になる。これにより電流のリミットを低めに設定しておくことでショート等による回路の損傷が防げる。また電流リミット機能を積極的に使うことで、定電流源として使う場合もある。

本実験では当初は電源電圧 9V、電流リミット 0.3A 程度に設定する。その後、回路上の問題がなければ電流リミットを 1.5A 程度に上げる。(電圧は 9V のままで良い。このモータドライバの最大電圧は 11V であることに注意)。

なお写真の電源は出力用のボタンがあるが、他の CVCC 電源（安い）では出力ボタンがなく、電源投入瞬間から出力されるので、電源投入前に電圧ボリュームを左に回しておく。

CVCC 電源については例えば下記で自習する。

http://www.kikusui.co.jp/knowledgeplaza/powersupply1/powersupply1_j.html

https://www.kikusui.co.jp/faq/topic.php?id=XOfJmMqF4OUx

図 26 のサンプルでモータが動くことを確認する。

```
const int MODE = 8; //GP8
const int M1_IN_A = 9; //GP9
const int M1_IN_B = 10; //GP10

int t=0;

void setup() {
  pinMode(MODE, OUTPUT);
  pinMode(M1_IN_A, OUTPUT);
  pinMode(M1_IN_B, OUTPUT);
  digitalWrite(MODE,LOW);
}

void loop() {
  t=(t+1)%2;

  if(t==0){
    digitalWrite(M1_IN_A,HIGH);
    digitalWrite(M1_IN_B,LOW);
  }else{
    digitalWrite(M1_IN_A,LOW);
    digitalWrite(M1_IN_B,HIGH);
  }
  delay(1000);
}
```

図 26　モータドライバを動かすサンプルプログラム

[課題17]　　Hブリッジ回路におけるストップとブレーキの違いを原理を含めて説明せよ。

図 25 のファンクションテーブルから、図 27 のような関数(motor)によって出力をアナログ的に制御できると考えられる。第 7 章で利用した PWM 出力をここでも利用している。

```
const int MODE = 8; //GP8
const int M1_IN_A = 9; //GP9
const int M1_IN_B = 10; //GP10

int t=0;

void motor(float p)// p must be -1.0 to 1.0
{
  if(p>0){
    analogWrite(M1_IN_A,0);
    analogWrite(M1_IN_B,(int)(p*65535.0));
  }else{
    analogWrite(M1_IN_B,0);
    analogWrite(M1_IN_A,(int)(-p*65535.0));
  }
}

void setup() {
  pinMode(MODE, OUTPUT);
  digitalWrite(MODE,LOW);
  analogWriteFreq(2000); //PWM set to 2kHz
  analogWriteResolution(16); //Analog range set to 0-65535
  Serial.begin (9600);
}

void loop() {
  t=(t+1)%200; //0 to 199
  motor((float)(t-100)/100.0); //-1.0 to 1.0
  Serial.println(t);
  delay(100);
}
```

図 27　H ブリッジ回路を PWM 駆動するサンプルプログラム

[課題18]　　シリアル通信で指令を送り、例えば f(orward)、b(ackward)を入力すると正転／逆転が切り替わり、h(igh)、l(ow)を入力するたびに速度が変更されるようにせよ。PWM の制御周波数を 50Hz、1kHz、20kHz と変化させ、振る舞いの変化、モータから発する音を観察し、考察せよ。20kHz に設定し、PWM 出力が予想通りの出力になっているかどうかをオシロスコープで計測して確認せよ。（おそらく 20kHz では駆動できていないはずで、指定ビット幅を減らす必要がある。これも試すこと）この後は 20kHz の設定で用いる。

[課題19]　　一方向に回転させ続けている時、電流はどの程度流れているか。またこのモータの回転を手で止めた時（気をつける！）、電流はどの程度流れるか。この違いはなぜ生じるか調べ、説明せよ（キーワード：DC モータ、逆起電力）。

9.3 PID 制御

エンコーダによる角度計測と PWM によるモータ駆動を組み合わせて、「目標位置に向かって動く」ロボットアームを作成する。また制御を変化させることでいろいろな力覚提示が可能となることを観察する。ここでは P 制御、PD 制御等を用いる。もしインタラクティブシステム論を受講していなければ下記などで自習する。

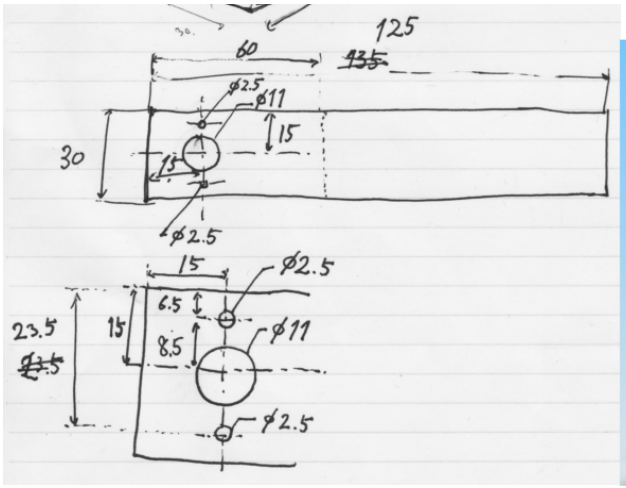
https://kaji-lab.jp/ja/index.php?plugin=attach&refer=people%2Fkaji%2Fninshiki&openfile=ninshiki2022_08.pdf

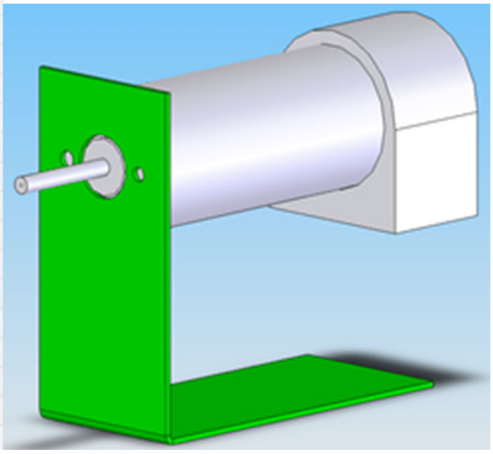
https://www.youtube.com/watch?v=KjLsA7YB2Yc

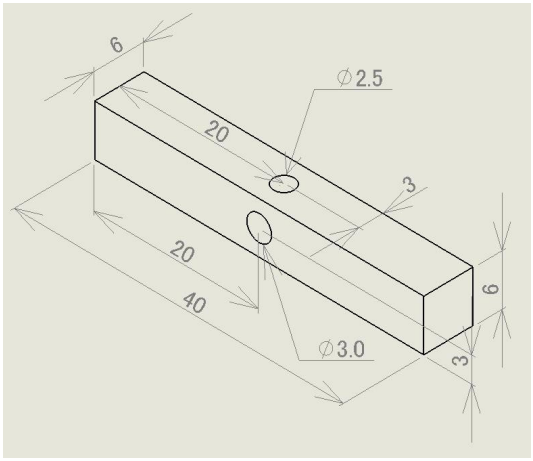
[課題20]　　アルミ加工により、DC モータの軸に取手をつける。（この部分は下記に従い個別に指導する）

アルミ加工の講習内容

1. ケガキの仕方
    - ① ノギスの使い方
    - ② ハイトゲージの使い方
    - ③ ポンチの打ち方
2. バンドソーの使い方
3. ドリルの使い方
    - ① 台座の緩め方、上下させ方
    - ② ドリルの交換方法
    - ③ 万力による材料の固定
    - ④ 穴あけ、小さい穴に関するバリとり
    - ⑤ 特殊ドリル１：ステップドリルの使い方
    - ⑥ 特殊ドリル２：バリ取り用ドリルの使い方
4. タップの切り方
5. 後片付け：掃除の仕方（工具周りを刷毛で。床を掃除機で）







これ以降はアルミ台座を机にガムテープで固定する（図 28）。

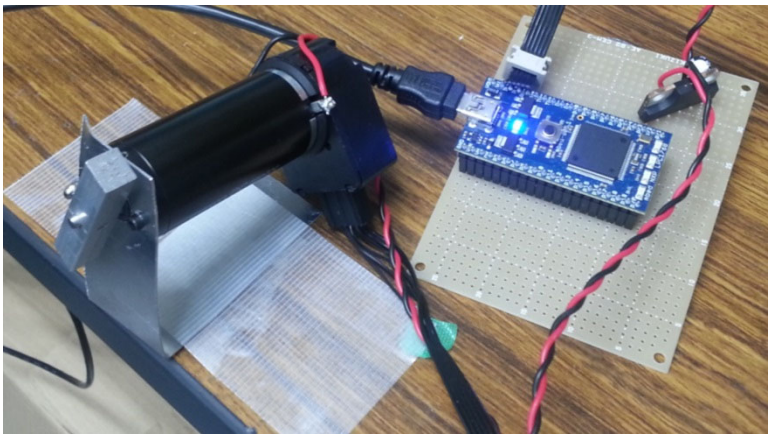


図 28　ノブ付きモータと制御基板

[課題21]　　P 制御により目的の位置（初期位置で良い）に向かうようにし、電源を付け、取手を手で回転させてみる。どうなるか。ゲインを上げ、なるべく早く戻るようにする。なおこのときのループ周期を計測し、常に 1kHz ループが維持されるようにする。この

ためにはタイマ割り込みを使っても、loop 関数中で適切な delay を挟んでも良い。この後も特に断りがない限り同様に計測する。

[課題22]　PD 制御により目的の位置（初期位置で良い）に向かうようにし、電源を付け、取手を手で回転させてみる。どうなるか。また PC から位置指令を送り（例えば r(ight), l(eft)）、キーボードを押すたびに移動するようにせよ。 なお D 成分の計算のためには微分が必要であるが、micros 関数などによりループ周期を計測してその時間⊿t を用いて計算する（ループ周期が変化したときの対処のため）

[課題23]　PID 制御により目的の軌道を描かせる。軌道としては例えば 1 周 1 秒の回転を行う。このときの PID 制御は、現在位置と目標軌道との差に対して行う。また PC から指令を送り（例えば r(ight), l(eft)）、キーボードを押すたびに回転方向が変わるようにする。

[課題24]　ある場所でぶつかる「壁」を表現する。こうした場合、ある角度を超えた角度を「めり込み量」と考え、このめり込み量に比例した反力を提示すればバネ性の壁が表現できる。この比例定数（ばね定数）を高くすればより硬い壁が表現できるはずである。なるべく高い比例定数を実現する。このときのループ周期を 100Hz、1kHz、10kHz とし、表現可能な硬さがどのように変化するかを調べる。

10 ステッピングモータ

DC モータは比較的制御が容易で高速なフィードバックにより力覚提示を実現することも容易であるが、指定位置に移動したり、指定速度で回転したりする目的ではステッピングモータが多く用いられる。例えば 3D プリンタのヘッド位置を制御しているのはステッピングモータである。この章ではステッピングモータを実際に動かしてみる。（なお精度があまり必要でなく、特に多回転でない場合は 6 章の RC サーボモータでも良い。目的に対して適切なアクチュエータを使う）

回路は 9 章のものを使う。モータはバイポーラステッピングモータ ST-42BYH1004 を用いる。**このモータの定格電圧は５V なのでモータ用電源の電圧に注意する**（USB 給電でも一応動かせるが大電流（1A 以上）が必要で USB ポートの故障にもつながるので外部電源が望ましい）。

https://akizukidenshi.com/catalog/g/gP-07600/

バイポーラ型ステッピングモータの駆動方法については例えば下記を参照。プログラムは図 29 参照（秋月のサンプルを改変）。このプログラムで二相駆動が実現していることを理解する。

https://monoist.itmedia.co.jp/mn/articles/1607/11/news011.html

https://www.youtube.com/watch?v=g18dOXJ1Lgc ←わかりやすい！

```
const int MODE = 8;            //GP8
const int M1_IN_PHASE = 9;     //GP9
const int M1_IN_ENABLE = 10;   //GP10
const int M2_IN_PHASE = 11;    //GP11
const int M2_IN_ENABLE = 12;   //GP12

void setup() {
  pinMode(MODE, OUTPUT);
  digitalWrite(MODE,HIGH); //Phase/Enableモード
  pinMode(M1_IN_PHASE, OUTPUT);
  pinMode(M1_IN_ENABLE, OUTPUT);
  pinMode(M2_IN_PHASE, OUTPUT);
  pinMode(M2_IN_ENABLE, OUTPUT);
  digitalWrite(M1_IN_ENABLE,HIGH);//ブレーキ解除
  digitalWrite(M2_IN_ENABLE,HIGH);//ブレーキ解除
}

void loop() {
  digitalWrite(M1_IN_PHASE,HIGH);//状態：M1=正/M2=逆
  delay(10);
  digitalWrite(M2_IN_PHASE,HIGH);//状態：M1=正/M2=正
  delay(10);
  digitalWrite(M1_IN_PHASE,LOW);//状態：M1=逆/M2=正
  delay(10);
  digitalWrite(M2_IN_PHASE,LOW);//状態：M1=逆/M2=逆
  delay(10);
}
```

図 29　ステッピングモータ駆動プログラム

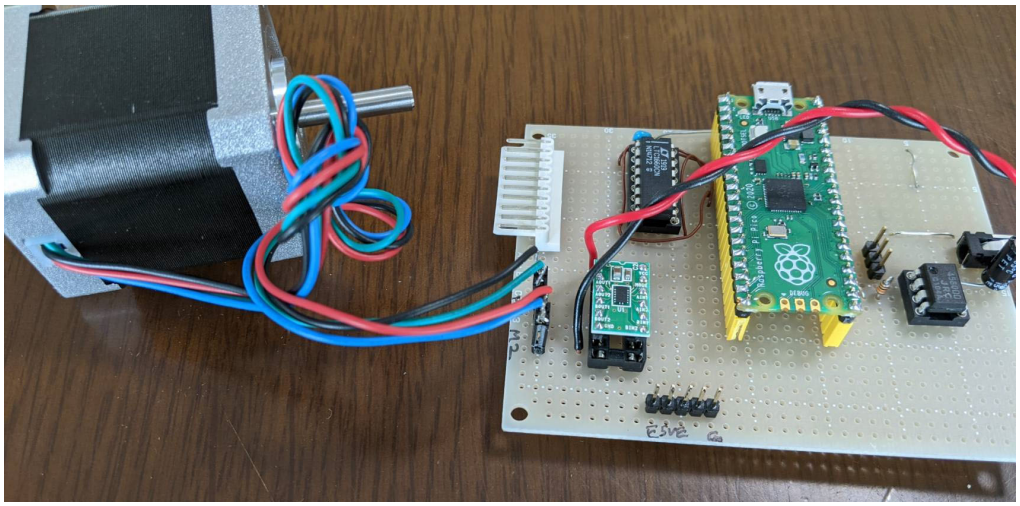


図 30　ステッピングモータ接続の様子

[課題25]　　上記のサンプルでは一方向にしか回転できない。回転方向を指定できるようにして PC からの通信によって実現する。ステッピングモータのスペック（ステップ角）から考えて適切な速度になっているか確認する。回転がわかりやすいように出力軸にテープを貼っておく。

[課題26]　　到達角度と速度を引数とする関数を作成する。角度に関しては正負の値が指定できるようにする。PC からのキーボード入力で位置指定をする。

なお講習会の範囲外であるが、ステッピングモータ特有の振動を低減する方法として「マイクロステップ駆動」が知られている。例えば下記のページを参照。
https://www.orientalmotor.co.jp/tech/webseminar/stkiso_2_2_2/
https://acetastronophen.blog.fc2.com/blog-entry-111.html