平成 27 年度

メカトロニクス演習 II テキスト

・組み込みコンピュータ（基礎・応用）

・位置制御実験

香川大学工学部

知能機械システム工学科

目　　次

メカトロニクス演習 II テキスト

II-1.組み込みコンピュータ

１．実験概要と目的

メカトロニクス機器，ロボットなどは，コンピュータを搭載し，制御されている。一般に，メカトロニクス機器などを構成する場合，配線部（コネクターを含む）は，エラーや故障の原因となることが多い。そのため，コンピュータを組み込み，配線部を短く，簡略することは，様々な機器を実用化する際に重要である。

本実験では，組み込みコンピュータとして，メカトロエンジニアに必須の電子回路と組み込みプログラミングの基礎を学ぶ。Arduino を利用した電子回路の組立とプログラミングを通じて，スイッチやセンサの入出力制御等，基本的な組み込み回路の構成とプログラム開発を理解することを目的とする。

２．実験準備

本実験では Arduino を使用して行う。Arduino は大きく 2 つの要素から成り立っており，ハードウェアである Arduino ボードと，ソフトウェアである Arduino IDE（コンピュータ上で動作する）を使用する。Arduino IDE は Arduino ボードにアップロードするためのスケッチ（プログラム）を作るために使用され，ボードに何をすべきかを伝える。

ソフトウェア（Arduino IDE）のインストール

〇以下は Windows8 を対象とした記述です。Windows のバージョンによって異なりますが，基本的な流れは同じですので，各項目のキーワードに注意して作業すること。

① 「arduino-1.6.5-r5-windows.exe」を各自 PC にコピーし，インストールする。

② Arduino インストール後，ボードと PC を USB ケーブルで接続する。

※Windows8 では，接続後自動的にドライバがインストールされる。Windows7 の場合 Arduino が自動認識されない場合があるので，※1 を参照して確認すること。

③ コントロールパネル＞システム＞デバイスマネージャーを開く（図 1-1)。ポート（COM と LPT）の中に Arduino Uno があるかを確認し，ポートの番号を確認しておく（図では COM3 に設定されている)。

④ インストールした Arduino を起動する。

⑤ ツール＞ポートを選択し，上記で確認したポート番号に設定する（図 1-2)。以上で設定は完了。

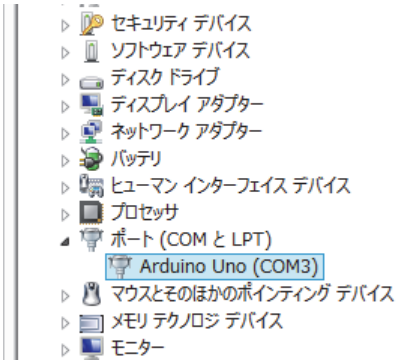


図 1-1 デバイスマネージャー画面（ポートは CM3）

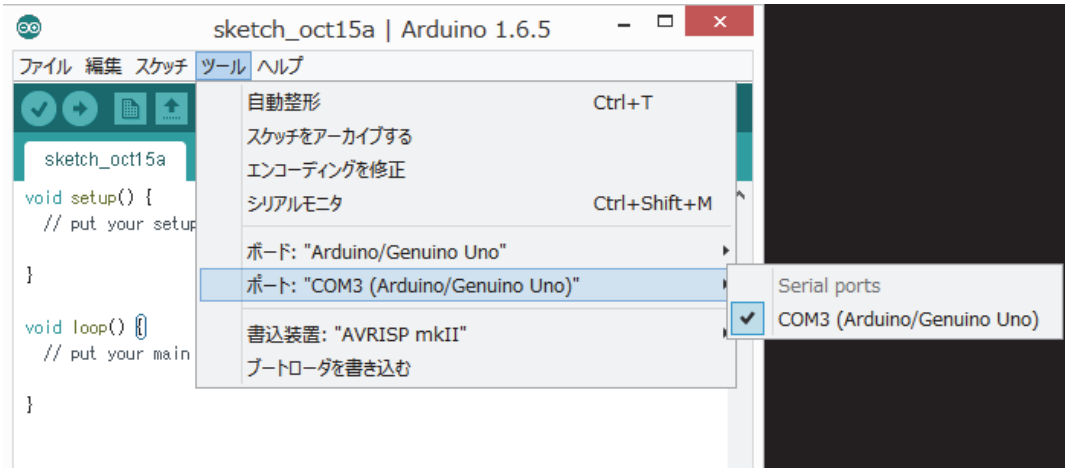


図 1-2　Arduino でのポート設定画面

※1　Windows7 等でデバイスが自動認識されない場合の対処方法

①　コントロールパネル＞システム＞デバイスマネージャーを開く（図 1-3）。

「Arduino Uno」を選択，右クリックから「ドライバの更新」を選択する。

※右クリックで「ドライバの更新」が表示されない場合は，「プロパティ」を選択。

表示されたプロパティの「ドライバの更新」を選択する。

「コンピュータを参照してドライバを検索する」を選択。

「参照」を選択し Arduino のインストールフォルダの中にある drivers フォルダを指定する（通常は「C->Program Files->Arduino->drivers」）。

※警告が出た場合も，無視してインストールを続ける。

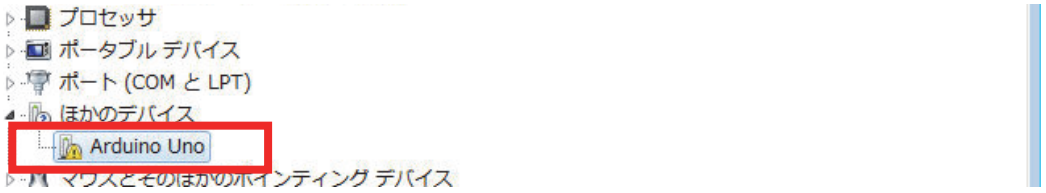


図 1-3　デバイスマネージャー画面（ポートが設定されていない）

② COM ポートが設定される（図 1-4 では COM15）ので確認しておく。

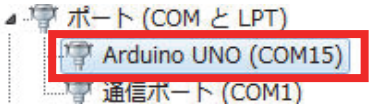


図 1-4　デバイスマネージャー画面

③ デバイスマネージャーを終了して，完了。
④ インストールした Arduino を起動する。
⑤ ツール＞ポートを選択し，上記で確認したポート番号に設定する。

※ 2 回目からの接続では，上記設定は不要。ただし，COM ポートは，ボードを取り付ける毎に設定されるため，USB ケーブルで接続する度に，デバイスマネージャー等を利用して，COM ポートを確認すること。

３．実験手順

(1)　Arduino キットについて

□Arduino キット

□Arduino Uno　×1
□ブレッドボード　×1
□Jumper Wire（L:100mm）　×10
□Parts Case　×1

□Jumper Wire (緑・L: 125mm)	×5	□Jumper Wire (黄・L:100 mm)	×5
□Jumper Wire (橙・L: 75mm)	×5	□Jumper Wire (赤・L: 50mm)	×5
□Jumper Wire (茶・L: 25mm)	×5	□Jumper Wire (白・L: 22.5mm)	×5
□Jumper Wire (緑・L: 20mm)	×5	□Jumper Wire (紺・L: 17.5mm)	×5
□Jumper Wire (青・L: 15mm)	×5	□Jumper Wire (緑・L: 10mm)	×5
□Jumper Wire (橙・L: 7.5mm)	×5	□Jumper Wire (赤・L: 5mm)	×5
□Jumper Wire (無・L: 2.5mm)	×5		
□タクトスイッチ	×1	□Cds センサ	×1
□10kΩ 抵抗	×2	□1kΩ 抵抗	×3
□330Ω 抵抗	×1	□LED(赤)	×1
□LED(緑)	×1	□LED(青)	×1

□Arduino 用モータードライバシールド「Ardumoto」　×1
□DC モータ　×2
□USB ケーブル　×1
□AC アダプター　×1

□USB メモリ　　×1

USB メモリ内部ファイル

arduino-0021

references

samples

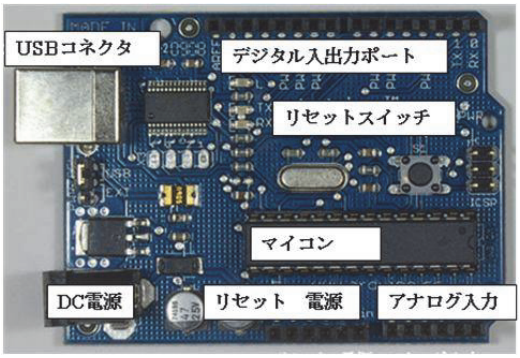


図 1-5　Arduino ボードレイアウト

(2)　Arduino 概要

2-1.　ボードレイアウト（図 1-5 参照）

2-2.　仕様

Microcontroller:	ATmega328
★Operating Voltage:	5V
Input Voltage (recommended):	7-12V
Input Voltage (limits):	6-20V
★Digital I/O Pins:	14 (of which 6 provide PWM output)　※1
★Analog Input Pins:	6　※2
DC Current per I/O Pin:	40 mA
DC Current for 3.3V Pin:	50 mA
Flash Memory:	32 KB of which 0.5 KB used by bootloader
SRAM:	2 KB
EEPROM:	1 KB
Clock Speed:	16 MHz

※1　14 本のデジタル I/O ピン（pin 0～13）

入力（INPUT）または出力（OUTPUT）として使用可。IDE で作成するスケッチのなかで，どちらかに設定する。また，このなかの 6 本（pin 3,5,6,9,10,11）はスケッチで設定することによりアナログ出力として使用可となる。

※2　6 本のアナログ IN ピン（pin 0～5）

アナログの値を得るために使用可（例：センサからの電圧の読み取り）。得られた値は 0～1023 までの数値に変換される。

✍電源について

Arduino ボードはコンピュータの USB ポート，USB 充電器，AC アダプター等を電源とすることができる。電源端子に何も接続されていないときは USB ポートから電源をとる。

(3) プログラミング手順

LED 点滅プログラムのサンプルを実行して, Arduino ボードと Arduino IDE が正常に動作するか確認する。LED（発行ダイオード）はアクチュエータとしてもっとも使われることが多い電子部品である。Arduino ボードには LED があらかじめ搭載されており，「L」のマークで示されている。もちろん，キットの中にある LED を自分で付け足すことも可能である。

3-1.　Blink プログラムによる動作確認

① Arduino IDE を起動する。※デスクトップ上にショートカットを作成しておくと便利。

② ファイル＞スケッチの例＞01.Basics＞Blink の順にファイルを開く（図 1-6)。

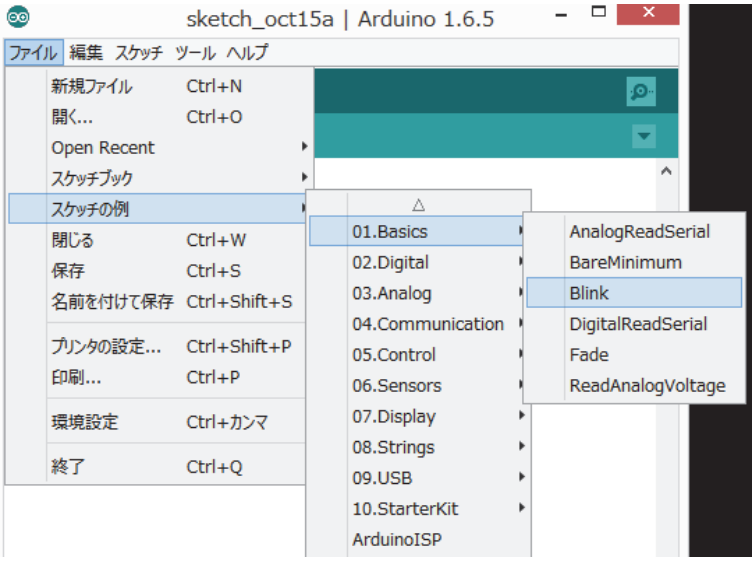


図 1-6　Blink ファイルの選択

③ ツール＞Board でボードの選択を行い Arduino Uno が選択されているか確認（図 1-7)。

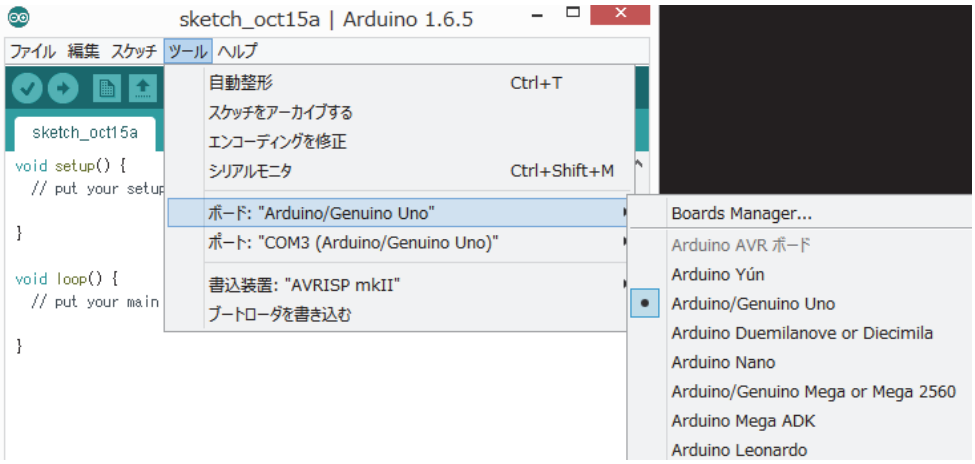


図 1-7　ボードの確認，選択

④ メニューバにある右向き矢印のアイコン（マイコンボードに書き込むためのアイコン）をクリックし，プログラムを書き込む（図 1-8）。

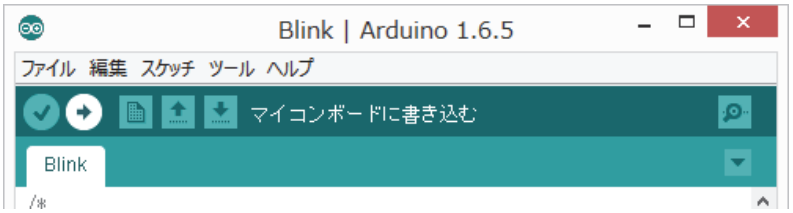


図 1-8　アップロードボタン

⑤ ウインドウ下部の黒い欄の上に，図 1-9 のように表示されれば書き込み成功。

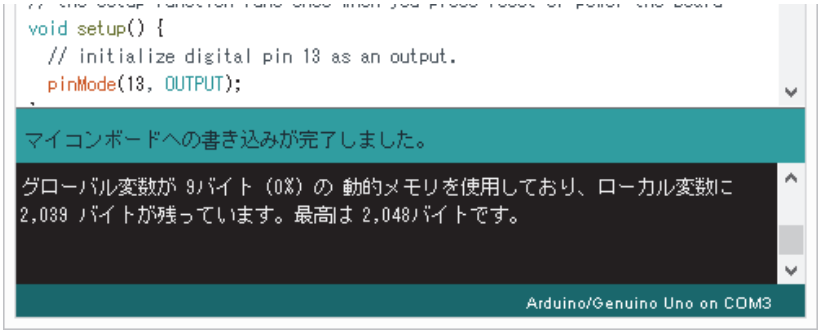


図 1-9　プログラムの書き込み成功画面

⑥ 書き込みが成功すると付属の LED が 1 秒間隔の点滅を開始する。

なお，自分でプログラムを新規作成する場合は，ファイル＞新規ファイルを開き，スケッチを行った後，ファイル＞名前を付けて保存，を行うこと。

いったん Arduino ボードにアップロードしたプログラムは，別のプログラムをアップロードするまでボード上に残る。リセットや電源オフによって消えることはない。

3-2. Arduino プログラムの基本構成

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

図 1-10　サンプルプログラム“Blink”

図 1-10 にサンプルプログラム“Blink”を示す。Arduino はプログラムを 1 行目から最終行に向かって順番に実行していくこと，また setup()と loop()の 2 つの関数が基本となっており，ともに省くことはできないことを覚えておくこと。基本的にはこの 2 つの関数に対して，必要なコードを追加して，プログラムを構成する。もちろん自分で関数を作ることもできる。C 言語同様，セミコロンを忘れないこと。なお，“//”以降はコメントである。

setup()

初期設定を行うための関数である。プログラムが動くときに，一度だけ実行したいコードを書く。例えば入出力ピンの設定，変数や定数の宣言など。

loop()

希望する動作のプログラムを記述する。一般的な C 言語のプログラムでは main()と同様の働きとなる。ただし，C 言語の main()などと違い，loop()に記述された命令は，自動的に繰り返される（図 1-11 参照）。Arduino では，プログラムを書き込んだ時点から，電源を切るまで，この loop()が無限に繰り返される。Arduino は止まらないことを覚えておくこと。

Arduino プログラムは，C/C++に準拠している。特に C 言語については，ほぼすべての構造を網羅しているので，違いを意識することなく，使用できる。C++については，公式リファレンスを参照されたい。

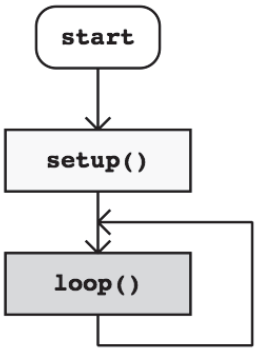


図 1-11　Arduino における処理のフローチャート

3-3.　Arduino プログラム例題

13 番ピンの LED を 2 秒間点灯させた後，0.5 秒間消灯させ，それを繰り返すプログラムを作成せよ。

【解答】

＜プログラム例>

```
    void setup() {                          //初期設定関数
          pinMode( 13, OUTPUT );            //13 番ピン出力に設定
    }

    void loop() {
          digitalWrite( 13, HIGH );         //LED を点灯
          delay( 2000 );                    //2 秒間ウエイト
          digitalWrite( 13, LOW );          //LED を消灯
          delay( 500 );                     //0.5 秒間ウエイト
    }
```

図 1-12　フローチャート例

【解説】

pinMode (pin 番号, mode)

入出力を伴う場合，setup()の中でピンの設定をする必要がある。ここでは LED をコントロールするために出力ピンが必要なので，pin 番号として 13 番，モードは OUTPUT と指定する。入力として使用する場合は INPUT とする。なお，pinMode は関数であり，関数の後ろの（）の中に置かれる言葉や数値を引数という。

digitalWrite (pin 番号, value)

出力に設定されたピンをオン（HIGH）またはオフ（LOW）にする。最初の引数（pin 番号）はどのピンを制御するか指定するためのもの，2 つ目の引数でピンをオンにするかオフにするかを指定している。指定したピンが OUTPUT に設定されている場合，HIGH = 5V，LOW = 0V (GND) にセットされる。INPUT に設定されている場合は，HIGH を出力すると 20KΩの内部プルアップ抵抗が有効に，LOW で無効にセットされる。

delay(2000)

引数で指定したミリ秒だけマイコンをストップさせる。ここで LED の変化は 2 秒間停止することになる。delay(500)は 0.5 秒。

４．演習（基礎）

以下の 4 課題について順にプログラミングする。各課題については，フローチャート（授業内にて説明する）を作成の上，プログラムのソースファイルを保存しておくこと。また，ブレッドボード上に回路を構成した場合は，回路の写真を見えやすいように撮影しておくこと。

【課題番号：1001】LED の点滅

13 番ピンの LED を，1 秒間点灯，1 秒消灯を 5 回繰り返した後，2 秒間点灯，2 秒間消灯を 3 回繰り返すルーチンを交互に実行するプログラムを作成せよ。

【課題番号：1002】LED のコントロール

タクトスイッチを押すと LED が点灯し，再度押すと消灯するプログラムを作成せよ。

【課題番号：1003】明るさのコントロール

通常時は LED が消灯している状態から，タクトスイッチを押し続けた場合に，LED が点灯し，明るさが増していくプログラムを作成せよ。ただし，スイッチから指を離したら，消灯すること。

【課題番号：1004】LED の高度なコントロール

タクトスイッチを二つ使い，LED を点灯させるプログラムを以下のルールに従って作成せよ。

（1） LED が消灯している状態で，タクトスイッチ A を押すと，LED が点灯する。

（2） LED が点灯している状態で，タクトスイッチ B を押すと，押している間，点滅し，タクトスイッチ B から指を離すと，点灯状態に戻る。

（3） LED が点灯している状態で，タクトスイッチ A を押すと，LED が消灯する。

５．演習（応用）

以下の 3 課題について順にプログラミングする。各課題についてプログラムのソースファイルを保存しておくこと（フローチャートは不要)。また，ブレッドボード上に回路を構成した場合は，回路の写真を見えやすいように撮影しておくこと。

【課題番号：2001】2 進数カウンタ

LED4 個とタクトスイッチ 1 個を用いて，2 進カウンタを作成せよ。

※ボタンを押すごとに，カウンタが進み，LED4 つで，2 進数 4 桁を表示するカウンタを作成する。

【課題番号：2002】光通信 1

二人で，光通信を行う回路ならびにプログラムを以下の条件にしたがって作成せよ。Arduino2 枚を A と B と異なった機能で実現する物とする。

（1） Arduino A を送信側とし，Arduino B を受信側とする。

（2） Arduino A は, タクトスイッチと LED から構成され, タクトスイッチを押している間, 1 秒毎に点滅する。

（3） Arduino B は, 受光素子と LED で構成され, Arduino A からの信号を受信している間, 0.2 秒毎に点滅する。

【課題番号：2003】光通信 2

Arduino A からの信号が，Arduino B に入った場合，Arduino B は，割り込みによって，その動作を停止するプログラムを作成せよ。

（1） Arduino A は, タクトスイッチと LED から構成され, タクトスイッチを押している間, 点滅し続ける。

（2） Arduino B は, 受光素子と LED で, 構成され, 通常時は, 0.5 秒毎に LED を点灯させ, Arduino A より，信号が入力された場合に，0.1 秒間隔で 5 回 LED を点滅させた後, 消灯する。

６．レポート作成要領

レポートはプログラミング演習基礎課題 1001～1004 及び応用課題 2001～2003 をまとめて提出してもらう。以下の点に注意してレポートを作成すること。

(1) 基礎課題 1001～1004 は，レポート提出必須課題である。この 4 課題のうち，1 つでも欠けている場合，提出として認めないので注意すること。また，基礎課題については，
・ソースプログラム
・フローチャート
・ブレッドボードを撮影した回路写真（ただし 1001 については不要）
の 3 点を必ず掲載すること。掲載していないものは減点対象となる。

(2) 応用課題 2001～2003 については，ソースプログラムと回路写真の掲載を必須とし，フローチャートは任意とする。

(3) プログラム作成，回路構築などで感じた問題や，実験を行った上での感想があれば，まとめておくこと。（任意）

参考資料

I. フローチャート記号

フローチャートとは流れ図のことで，各作業のステップを下記のような図形で表し，それらの間を実線や矢印でつないで流を表すものである。これにより，プログラムの設計においてアルゴリズムやプロセスを表現できる。

プログラムにおいては，実際にプログラムを組む前の段階でフローチャートを作成することで，

・図で示すことで，文章より分かりやすくなる。
・プログラムを改良する際に効率よくできる。
・プログラムの理解の手助けとなる（第 3 者に伝わりやすい)。
・複雑なプログラムを複数人で組む際，手分けしやすい。

フローチャートを作成する際の基本事項は以下のとおりである。

(1) フローの最初と最後を明記する。
(2) 処理の流れは原則として，上から下へ，もしくは左から右へ。それに逆行する際には矢印をつけること。
(3) 線が交差しないようにすること。

フローチャートの記号は，日本工業規格（JIS）で決められている。主な種類を図 1-13 に示す。

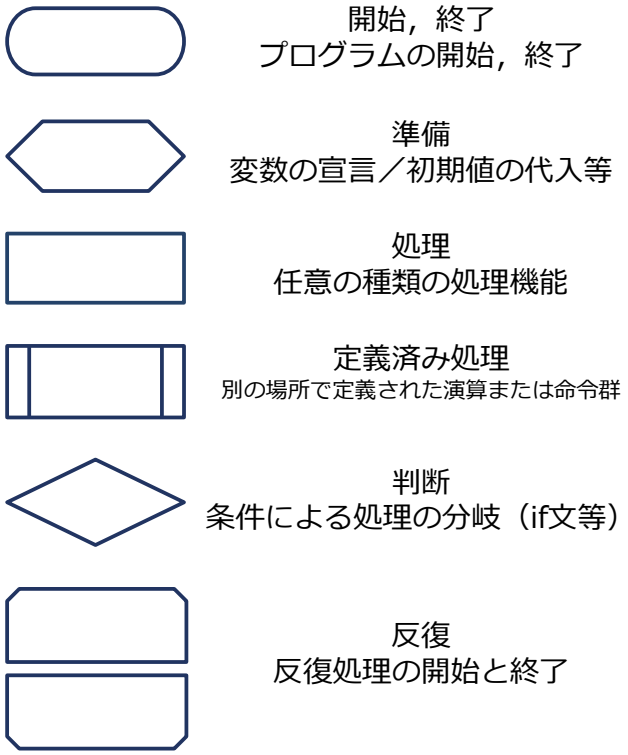


図 1-13　フローチャートの主な記号

II. Arduino 関数リファレンス

A. 制御文

[1] if (conditional) and ==, !=, <, > (comparison operators)

if, which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an if test is:

```
if (someVariable > 50) {
      // do something here
}
```

The program tests to see if some variable is greater than 50. If it is, the program takes a particular action. Put another way, if the statement in parentheses is true, the statements inside the brackets are run. If not, the program skips over the code.
The brackets may be omitted after an *if* statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
      digitalWrite(LEDpin1, HIGH);
      digitalWrite(LEDpin2, HIGH);
}                                                    // all are correct
```

The statements being evaluated inside the parentheses require the use of one or more operators:

Comparison Operators:
x = = y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

[2] if else

if/else allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500){
      // action A
}
else{
      // action B
}
```

else can proceed another **if** test, so that multiple, mutually exclusive tests can be run at the same time.
Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default **else** block is executed, if one is present, and sets the default behavior.
Note that an **else if** block may be used with or without a terminating **else** block and vice versa. An unlimited number of such **else if** branches is allowed.

```
if (pinFiveInput < 500) {
      // do Thing A
}
else if (pinFiveInput >= 1000){
      // do Thing B
}
else{
      // do Thing C
}
```

Another way to express branching, mutually exclusive tests, is with the switch case statement.

[3] switch / case statements

Like **if** statements, **switch...case** controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.
The **break** keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Syntax

```
switch (var) {
        case label:
        // statements
        break;
        case label:
        // statements
        break;
        default:
        // statements
}
```

Example

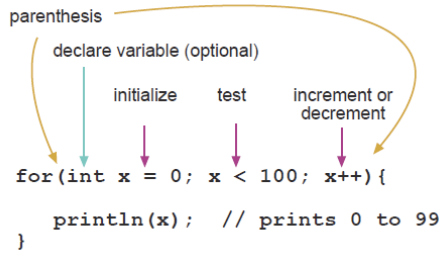
```
switch (var) {
        case 1:
        //do something when var equals 1
        break;
        case 2:
        //do something when var equals 2
        break;
        default:
        // if nothing else matches, do the default
        // default is optional
}
```

Parameters
var: the variable whose value to compare to the various cases
label: a value to compare the variable to

[4] for statements

The **for** statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.
There are three parts to the **for** loop header:

for (**initialization**; **condition**; **increment**) {
//statement(s);
}



The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Example

```
// Dim an LED using a PWM pin
int PWMpin = 10;                    // LED in series with 470 ohm resistor on pin 10

void setup() {
        // no setup needed
}

void loop() {
        for (int i=0; i <= 255; i++){
                analogWrite(PWMpin, i);
                delay(10);
        }
}
```

Coding Tips
The C **for** loop is much more flexible than **for** loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and increment can be any valid C statements with unrelated variables, and use any C datatypes including floats. These types of unusual **for** statements may provide solutions to some rare programming problems.
For example, using a multiplication in the increment line will generate a logarithmic progression:

```
for(int x = 2; x < 100; x = x * 1.5) {
        println(x);
```

```
}
```

Generates: 2,3,4,6,9,13,19,28,42,63,94

Another example, fade an LED up and down with one **for** loop:

```
void loop() {
    int x = 1;
    for (int i = 0; i > -1; i = i + x){
        analogWrite(PWMpin, i);
        if (i = 255) x = -1;                    // switch direction at peak
        delay(10);
    }
}
```

[5] while loops

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the **while** loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax
```
while(expression) {
// statement(s)
}
```
Parameters expression - a (boolean) C statement that evaluates to true or false

Example
```
var = 0;
while(var < 200) {                              // do something repetitive 200 times
    var++;
}
```

[6] do - while

The **do** loop works in the same manner as the **while** loop, with the exception that the condition is tested at the end of the loop, so the **do** loop will *always* run at least once.

```
do {
    // statement block
} while (test condition);
```

Example
```
do {
    delay(50);                                  // wait for sensors to stabilize
    x = readSensors();                          // check the sensors
} while (x < 100);
```

[7] break

break is used to exit from a **do**, **for**, or **while** loop, bypassing the normal loop condition. It is also used to exit from a **switch** statement.

Example
```
for (x = 0; x < 255; x ++) {
    digitalWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){                      // bail out on sensor detect
        x = 0;
        break;
    }
    delay(50);
}
```

[8] continue

The continue statement skips the rest of the current iteration of a loop (**do**, **for**, or **while**). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

Example
```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){                     // create jump in values
        continue;
    }
    digitalWrite(PWMpin, x);
    delay(50);
}
```

[9] return

Terminate a function and return a value from a function to the calling function, if desired.

Syntax:
```
return;
return value; // both forms are valid
```
Parameters value: any variable or constant type

Examples:

```
A function to compare a sensor input to a threshold
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    else{
        return 0;
    }
}
```

The return keyword is handy to test a section of code without having to "comment out" large sections of possibly buggy code.

```
void loop(){
    // brilliant code idea to test here
    return;
    // the rest of a dysfunctional sketch here
    // this code will never be executed
}
```

[10] goto

Transfers program flow to a labeled point in the program

Syntax label:
goto label; // sends program flow to the label

Tip
The use of *goto* is discouraged in C programming, and some authors of C programming books claim that the *goto* statement is never necessary, but used judiciously, it can simplify certain programs. The reason that many programmers frown upon the use of *goto* is that with the unrestrained use of *goto* statements, it is easy to create a program with undefined program flow, which can never be debugged.
With that said, there are instances where a goto statement can come in handy, and simplify coding. One of these situations is to break out of deeply nested *for* loops, or *if* logic blocks, on a certain condition.

Example

```
for(byte r = 0; r < 255; r++){
    for(byte g = 255; g > -1; g--){
        for(byte b = 0; b < 255; b++){
            if (analogRead(0) > 250){ goto bailout;}
               // more statements ...
        }
    }
}
bailout:
```

B. デジタル入出力関数

[11] pinMode()

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details.

Syntax pinMode(pin, mode)
Parameters pin: the number of the pin whose mode you wish to set
mode: either INPUT or OUTPUT
Returns None

Example

```
int ledPin = 13;   // LED connected to digital pin 13

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

void loop() {
    digitalWrite(ledPin, HIGH);         // sets the LED on
    delay(1000);                          // waits for a second
    digitalWrite(ledPin, LOW);          // sets the LED off
    delay(1000);                        // waits for a second
}
```

NOTE: The analog input pins can be used as digital pins, referred to as A0, A1, etc.

[12] digitalWrite()

Write a HIGH or a LOW value to a digital pin.
If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.
If the pin is configured as an INPUT, writing a HIGH value with digitalWrite() will enable an internal 20K pullup resistor (see the tutorial on digital pins). Writing LOW will disable the pullup. The pullup resistor is enough to light an LED dimly, so if LEDs appear to work, but very dimly, this is a likely cause. The remedy is to set the pin to an output with the pinMode() function.

NOTE: Digital pin 13 is harder to use as a digital input than the other digital pins because it has an LED and resistor attached to it that's soldered to the board on most boards. If you enable its internal 20k pull-up resistor, it will hang at around

1.7 V instead of the expected 5V because the onboard LED and series resistor pull the voltage level down, meaning it always returns LOW. If you must use pin 13 as a digital input, use an external pull down resistor.

Syntax digitalWrite(pin, value)
Parameters pin: the pin number
Value: HIGH or LOW
Returns none

Example

```
int ledPin = 13;                        // LED connected to digital pin 13

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

void loop() {
      digitalWrite(ledPin, HIGH);       // sets the LED on
      delay(1000);                      // waits for a second
      digitalWrite(ledPin, LOW);        // sets the LED off
      delay(1000);                      // waits for a second
}
```

Sets pin 13 to HIGH, makes a one-second-long delay, and sets the pin back to LOW.

NOTE: The analog input pins can be used as digital pins, referred to as A0, A1, etc.

[13] digitalRead()

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax digitalRead(pin)
Parameters pin: the number of the digital pin you want to read (*int)*
Returns HIGH or LOW

Example

```
int ledPin = 13;                        // LED connected to digital pin 13
int inPin = 7;                          // pushbutton connected to digital pin 7
int val = 0;        // variable to store the read value

void setup() {
      pinMode(ledPin, OUTPUT);          // sets the digital pin 13 as output
      pinMode(inPin, INPUT);             // sets the digital pin 7 as input
}

void loop() {
      val = digitalRead(inPin);          // read the input pin
      digitalWrite(ledPin, val);         // sets the LED to the button's value
}
```

Sets pin 13 to the same value as the pin 7, which is an input.

NOTE: If the pin isn't connected to anything, digitalRead() can return either HIGH or LOW (and this can change randomly). The analog input pins can be used as digital pins, referred to as A0, A1, etc.

C. アナログ入出力関数

[14] analogRead()

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().
It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax analogRead(pin)
Parameters pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)
Returns int (0 to 1023)

NOTE: If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

Example

```
int analogPin = 3;                      // potentiometer wiper (middle terminal) connected to
                                        // analog pin 3, outside leads to ground and +5V
int val = 0;        // variable to store the value read

void setup() {
      Serial.begin(9600);               // setup serial
}

void loop() {
      val = analogRead(analogPin);      // read the input pin
      Serial.println(val);              // debug value
}
```

[15] analogWrite()

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal is approximately 490 Hz.
On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 through 13. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11. You do not need to call pinMode() to set the pin as an output before calling analogWrite().
The *analogWrite* function has nothing whatsoever to do with the analog pins or the *analogRead* function.

Syntax analogWrite(pin, value)
Parameters pin: the pin to write to.
value: the duty cycle: between 0 (always off) and 255 (always on).
Returns nothing

Notes and Known Issues
The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g 0 - 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6.

Example
Sets the output to the LED proportional to the value read from the potentiometer.

```
int ledPin = 9;                          // LED connected to digital pin 9
int analogPin = 3;                       // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

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

void loop() {
      val = analogRead(analogPin);       // read the input pin
      analogWrite(ledPin, val / 4);      // analogRead values go from 0 to 1023,
}                                        // analogWrite values from 0 to 255
```

[16] analogReference(type)

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:

DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (*not available on the Arduino Mega*)
INTERNAL1V1: a built-in 1.1V reference (*Arduino Mega only*)
INTERNAL2V56: a built-in 2.56V reference (*Arduino Mega only*)
EXTERNAL: the voltage applied to the AREF pin is used as the reference.

Parameters type: which type of reference to use (DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, or EXTERNAL).
Returns None.

D. その他の入出力関数

[17] shiftOut()

Shifts out a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed to indicate that the bit is available.
This is a software implementation; Arduino (as of 0019) also provides an SPI library that uses the hardware implementation.

Syntax shiftOut(dataPin, clockPin, bitOrder, value)
Parameters dataPin: the pin on which to output each bit (int)
clockPin: the pin to toggle once the dataPin has been set to the correct value (*int*)
bitOrder: which order to shift out the bits; either MSBFIRST or LSBFIRST.
(Most Significant Bit First, or, Least Significant Bit First)
value: the data to shift out. (*byte*)
Returns None

NOTE: The **dataPin** and **clockPin** must already be configured as outputs by a call to pinMode().
shiftOut is currently written to output 1 byte (8 bits) so it requires a two step operation to output values larger than 255.

```
// Do this for MSBFIRST serial
int data = 500;
// shift out highbyte
shiftOut(dataPin, clock, MSBFIRST, (data >> 8));
// shift out lowbyte
shiftOut(data, clock, MSBFIRST, data);

// Or do this for LSBFIRST serial
data = 500;
// shift out lowbyte
shiftOut(dataPin, clock, LSBFIRST, data);
// shift out highbyte
shiftOut(dataPin, clock, LSBFIRST, (data >> 8));
```

Example *For accompanying circuit, see the* tutorial on controlling a 74HC595 shift register.

```
int latchPin = 8;   //Pin connected to ST_CP of 74HC595
int clockPin = 12;                          //Pin connected to SH_CP of 74HC595
int dataPin = 11;  ////Pin connected to DS of 74HC595

void setup() {
      //set pins to output because they are addressed in the main loop
      pinMode(latchPin, OUTPUT);
      pinMode(clockPin, OUTPUT);
      pinMode(dataPin, OUTPUT);
}

void loop() {
      //count up routine
      for (int j = 0; j < 256; j++) {
            //ground latchPin and hold low for as long as you are transmitting
            digitalWrite(latchPin, LOW);
            shiftOut(dataPin, clockPin, LSBFIRST, j);
            //return the latch pin high to signal chip that it
            //no longer needs to listen for information
            digitalWrite(latchPin, HIGH);
            delay(1000);
      }
}
```

[18] pulseIn()

Reads a pulse (either HIGH or LOW) on a pin. For example, if **value** is **HIGH**, **pulseIn()** waits for the pin to go **HIGH**, starts timing, then waits for the pin to go **LOW** and stops timing. Returns the length of the pulse in microseconds. Gives up and returns 0 if no pulse starts within a specified time out.
The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax pulseIn(pin, value)
pulseIn(pin, value, timeout)
Returns the length of the pulse (in microseconds) or 0 if no pulse started before the timeout (*unsigned long*)
Parameters pin: the number of the pin on which you want to read the pulse. (*int*)
value: type of pulse to read: either HIGH or LOW. (*int*)
timeout (optional): the number of microseconds to wait for the pulse to start; default is one second (*unsigned long*)

Example

```
int pin = 7;
unsigned long duration;

void setup() {
      pinMode(pin, INPUT);
}

void loop() {
      duration = pulseIn(pin, HIGH);
}
```

[19] tone()

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.
Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.
Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

NOTE: if you want to play different pitches on multiple pins, you need to call noTone() on one pin before calling tone() on the next pin.

Syntax tone(pin, frequency)
tone(pin, frequency, duration)
Parameters pin: the pin on which to generate the tone
frequency: the frequency of the tone in hertz
duration: the duration of the tone in milliseconds (optional)
Returns Nothing

[20] noTone()

Stops the generation of a square wave triggered by tone(). Has no effect if no tone is being generated.

NOTE: if you want to play different pitches on multiple pins, you need to call noTone() on one pin before calling tone() on the next pin.

Syntax noTone(pin)
Parameters pin: the pin on which to stop generating the tone
Returns nothing

E. 時間に関する関数

[21] millis()

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Parameters None
Returns Number of milliseconds since the program started (*unsigned long*)

Example

```
unsigned long time;

void setup() {
    Serial.begin(9600);
}

void loop(){
    Serial.print("Time: ");
    time = millis();
    //prints time since program started
    Serial.println(time);
    // wait a second so as not to send massive amounts of data
    delay(1000);
}
```

Tip:
Note that the parameter for millis is an unsigned long, errors may be generated if a programmer tries to do math with other datatypes such as ints.

[22] micros()

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Duemilanove and Nano), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four). On 8 MHz Arduino boards (e.g. the LilyPad), this function has a resolution of eight microseconds.
Note: there are 1,000 microseconds in a millisecond and 1,000,000 microseconds in a second.

Parameters None
Returns Number of microseconds since the program started (*unsigned long*)

Example

```
unsigned long time;

void setup(){
    Serial.begin(9600);
}

void loop(){
    Serial.print("Time: ");
    time = micros();
    //prints time since program started
    Serial.println(time);
    // wait a second so as not to send massive amounts of data
    delay(1000);
}
```

[23] delay()

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax delay(ms)
Parameters ms: the number of milliseconds to pause (*unsigned long)*
Returns Nothing

Example

```
int ledPin = 13;                    // LED connected to digital pin 13

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

void loop() {
    digitalWrite(ledPin, HIGH);     // sets the LED on
    delay(1000);                    // waits for a second
    digitalWrite(ledPin, LOW);      // sets the LED off
    delay(1000);                    // waits for a second
}
```

Caveat
While it is easy to create a blinking LED with the delay() function, and many sketches use short delays for such tasks as switch debouncing, the use of delay() in a sketch has significant drawbacks. No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt. For alternative approaches to controlling timing see the millis() function and the sketch sited below. More knowledgeable

programmers usually avoid the use of delay() for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.
Certain things *do* go on while the delay() function is controlling the Atmega chip however, because the delay function does not disable interrupts. Serial communication that appears at the RX pin is recorded, PWM (analogWrite) values and pin states are maintained, and interrupts will work as they should.

[24] delayMicroseconds()

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.
Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax delayMicroseconds(us)
Parameters us: the number of microseconds to pause (*unsigned int)*
Returns None

Example

```
int outPin = 8;                         // digital pin 8

void setup() {
      pinMode(outPin, OUTPUT);          // sets the digital pin as output
}

void loop() {
      digitalWrite(outPin, HIGH);       // sets the pin on
      delayMicroseconds(50);            // pauses for 50 microseconds
      digitalWrite(outPin, LOW);        // sets the pin off
      delayMicroseconds(50);            // pauses for 50 microseconds
}
```

configures pin number 8 to work as an output pin. It sends a train of pulses with 100 microseconds period.

Caveats and Known Issues
This function works very accurately in the range 3 microseconds and up. We cannot assure that delayMicroseconds will perform precisely for smaller delay-times.
As of Arduino 0018, delayMicroseconds() no longer disables interrupts.

F. 外部割り込み

[25] attachInterrupt(interrupt, function, mode)

Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). The Arduino Mega has an additional four: numbers 2 (pin 21), 3 (pin 20), 4 (pin 19), and 5 (pin 18).

[26] detachInterrupt(interrupt)

Turns off the given interrupt.

G. 割り込み

[27] interrupts()

Re-enables interrupts (after they've been disabled by noInterrupts()). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

[28] noInterrupts()

Disables interrupts (you can re-enable them with interrupts()). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

II-2.位置制御実験

１．実験概要と目的

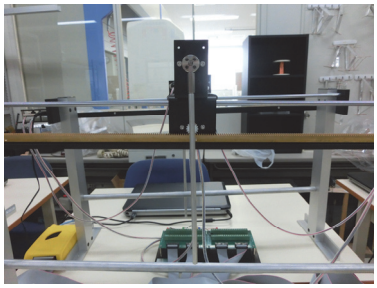
ロボットをはじめとするメカトロニクス機器において，高度な動作を実現している背景には，制御技術がある。制御とは，物体の運動を意図した位置や速度で動かす技術である。

精度の高い制御を行うためには，正しく制御理論を理解した上に，物体の運動を正しく解析し，モデル化する技術や，制御を行うためのパラメータの同定方法を身につける必要がある。

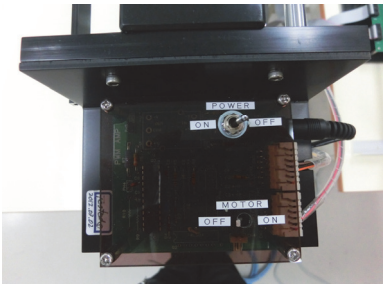
本実験では，制御技術を知る第一歩として，位置制御を例に，制御技術を体感することを目的とし，今後学習する制御理論，現代制御理論等を理解する上での基礎を養う。

２．実験装置概要

(1) 装置構成



a-1）実験装置本体



a-2）パワーアンプ

a-3)インターフェース（USB2）

a.直動型倒立振子実験装置

b.制御用コンピュータ

c.説明書

d.使用ソフトウェア

・直動型倒立振子制御プログラム

・OpenOffice

図 2-1　実験装置構成

(2) 操作手順

(2-1) 実験手順

① パワーアンプ部（モータ SW，電源 SW），USB2 のスイッチが OFF になっていることを確認する。

② PC を起動，USB2 のスイッチを ON にする。

③ PC が起動したら，デスクトップ上の STC01 をダブルクリックで，起動する。

④ 操作画面の右上の実験の枠内で「モータ角度テスト」が選択されていることを確認する。（選択されていない場合は，左側のラジオボタンをクリックして下さい。）

⑤ パワーアンプ電源 SW を ON にする。

⑥ 操作画面のパラメータを設定する。

⑦ 台車の振り子が外れていることを確認，または，取り外す。

⑧ 操作画面右上の START をクリックする。台車を手で動かしたときに，操作画面のモータ角度の数値が台車の移動に連動して変化することを確認する。（変化しない場合には，④から再度行い，それでも，動作しない場合には，①からやり直す。）

⑨ モータ角度を 0 近くになるまで，手で台車を動かす。

⑩ 台車を手で押さえて，パワーアンプモータ SW を ON にする。台車から手を離すと，モータ角度が 0 点付近まで，自動的に移動します。

⑪ 台車を少し，手で動かしてから，台車を話すと，0 点に戻ることを確認する。

⑫ 移動させるときは, 操作画面の Target に, 目標の数値(-15～15)を入力し, キーボードの ENTER キーをタイプするか，操作画面の START をクリックする。

(2-2) 終了手順

① Target に 0 を入力し，操作画面の STOP をクリックする。

② 操作画面の EXIT をクリック，もしくは，右上の X をクリックする。

③ パワーアンプ上で，モータ SW，電源 SW の順に OFF にする。

④ USB2 の電源を OFF にする。

⑤ PC をシャットダウンする。

(2-3) データの保存方法

① 操作画面が立ち上がったら，データの保存を設定する。

② 実験後生成された Plots.csv ファイルのコピーをデスクトップに作成，ファイル名を変更しておく。

③ OpenOffice（個人の PC へデータをコピーして使う場合は，CSV ファイルを読み取れるソフトウェア（Microsoft Excel 等）であれば可）にて,データを開き，グラフを作成する。（グラフの作成方法については，各自学習すること。）

３．実験手順

(1) 制御モデルの獲得

（1-1）運動方程式を求める。

台車の運動方程式は,

$$F = M\ddot{x} \tag{1}$$

である。ここでは，台車にかかる力をF，台車の質量をM として，移動量をxとしている。ただし，ラック部の損失は無視する。

モータの電圧vと回転角度θには，次の関係が成り立つ。

$$v = IR + K_m\dot{\theta} \tag{2}$$

式(2)において，Iはモータに流れる電流，Rはモータのコイル抵抗，K_mは，モータトルク定数である。また, モータの回転角度θと台車の移動量xの間には, 次の関係がある。ここで, rはモータギアの半径である。

$$\theta = \frac{x}{r} \tag{3}$$

モータから発生し，台車にかかるトルクは,

$$T = Fr = K_m I \tag{4}$$

で与えられる。

式(1)～(4)より，I，F，θを消去して，xに関する微分方程式とすると，次の様になる。

$$M\ddot{x} + \frac{{K_m}^2}{r^2 R}\dot{x} = \frac{K_m}{rR}v \tag{5}$$

（1-2）伝達関数を求める。

式(5)をラプラス変換する。（補足 I 参照）ただし，x，vのいずれも初期値はいずれも 0 であるとし，$L[x] = X$，$L[v] = V$とする。

$$s^2 MX + s\frac{{K_m}^2}{r^2 R}X = \frac{K_m}{rR}V \tag{6}$$

式(6)より,伝達関数$G(s) = \frac{X}{V}$を求めると，次の様になる。

$$G(s) = \frac{X}{V} = \frac{1}{\frac{K_m}{r}s + \frac{rRM}{K_m}s^2} \tag{7}$$

（1-3）PD 制御系を構築する。

制御における代表的かつ基本的な手法として，PD 制御が上げられる。PD 制御は，目標値と

出力値の差分（偏差）に対して比例するゲインと，偏差の微分値に比例するゲインを考えて制御入力を決定する手法である。

式(7)で求めた伝達関数を有するシステムに対して，目標値を x_t とした PD 制御のクローズドループを施したブロック線図は，図 2-2 で表される。ここで，K_p は比例ゲイン，K_d は微分ゲインを表している。

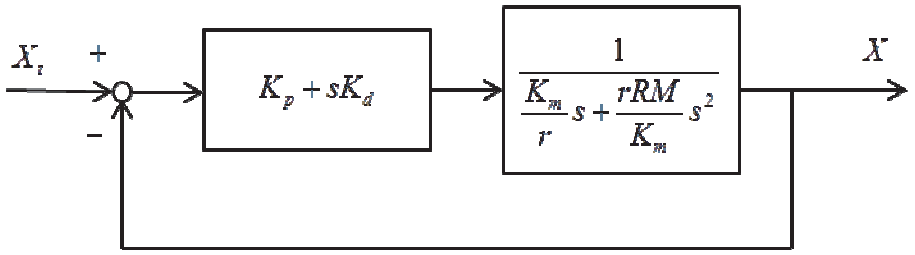


図 2-2　PD 制御系のブロック線図

図 2-2 で表されたクローズドループ伝達関数は，

$$\frac{X}{X_t} = \frac{G}{1+G} = \frac{(K_p + sK_d)\dfrac{1}{\dfrac{K_m}{r}s + \dfrac{rRM}{K_m}s^2}}{1 + (K_p + sK_d)\dfrac{1}{\dfrac{K_m}{r}s + \dfrac{rRM}{K_m}s^2}} = \frac{K_p + sK_d}{\dfrac{rRM}{K_m}s^2 + (\dfrac{K_m}{r} + K_d)s + K_p} \tag{8}$$

となる。

(2) 物理パラメータを同定する。

（2-1）操作手順を参考に，以下のとおり，ゲインを変更して，実験を行う。

はじめに，ゲインを，

$$K_p = 1,\ K_d = 0 \tag{9}$$

に設定し，静止した状態で，Target の数値に 5 を入れて，START をクリックする。生成された Plot.csv をデスクトップにコピーし，ファイル名を data_01.csv に変更する。

（2-2）実験で得られたデータより，ステップ応答のグラフを作成する。

前項の実験で保存した data_01.csv より，グラフを作成する。

（2-3）モータトルク定数を求める。

設定したゲインを，式（8）に入力すると，

$$\frac{X}{X_t} = \frac{1}{\frac{rRM}{K_m}s^2 + \frac{K_m}{r}s + 1} \tag{10}$$

となる。これは，2 次遅れ系（補足 II 参照）と呼ばれ，そのステップ応答では，減衰する周期 T_r が，

$$T_r = \frac{\pi}{\omega_n\sqrt{1-\delta^2}} = \frac{\pi r^2 RM}{\sqrt{K_m}\sqrt{r^3 RM - {K_m}^3}} \tag{11}$$

と求まる。（※減衰周期の求め方参照）そこで，（2-2）で作成したグラフより，減衰のピークの現れる時間を読み取れるだけ読み取り，その差の平均 $\overline{T_r}$ を求める。

式（11）に，減衰周期 $\overline{T_r}$ に加え，既知のパラメータ

台車の質量：　$M = 0.42 \quad kg$

モータギアの半径：　$r = 0.0063 \quad m$

モータコイル抵抗：　$R = 8.3 \quad \Omega$

を入力することで，モータトルク定数を求めることができる。

※減衰周期の求め方

式（10）で表された伝達関数の分母分子に $\frac{K_m}{rRM}$ をかけると，

$$\frac{X}{X_t} = \frac{1}{\frac{rRM}{K_m}s^2 + \frac{K_m}{r}s + 1} = \frac{\frac{K_m}{rRM}}{s^2 + \frac{{K_m}^2}{r^2 RM}s + \frac{K_m}{rRM}} \tag{12}$$

が得られる。これと，補足 II の式（2-1）を比較すると，

$${\omega_n}^2 = \frac{K_m}{rRM} \qquad \therefore \quad \omega_n = \sqrt{\frac{K_m}{rRM}} \tag{13}$$

$$\delta = \frac{{K_m}^2}{r^2 RM} * \frac{1}{\omega_n} = \frac{{K_m}^2}{r^2 RM} * \sqrt{\frac{rRM}{K_m}} = \sqrt{\frac{{K_m}^3}{r^3 RM}} \tag{14}$$

と求まる。これにより，減衰周期は

$$T_r = \frac{\pi}{\omega_n\sqrt{1-\delta^2}} = \frac{\pi}{\sqrt{\frac{K_m}{rRM}}\sqrt{1-\sqrt{\frac{{K_m}^3}{r^3RM}}^2}} = \frac{\pi r^2 RM}{\sqrt{K_m}\sqrt{r^3RM-{K_m}^3}} \qquad (15)$$

となる。

(3) 最適なゲインを試行錯誤的に求める。

（3-1）操作手順を参考に，以下の目標に対して，適切と思われるゲインを実験から求めよ。

（I） 収束を早くする。

（II） オーバーシュートを小さくする。

４．レポート作成要領

（1）レポートは，各週作成すること。

（2）第 1 週目は，モータトルク定数を求めるとこまで，計算過程をすべて明らかにして，求めよ。

（3）第 2 週目は，第 1 週目のレポートを持参すること。授業開始時にレポートのチェックを受けた後に，実験手順（3）を実施，実験結果に基づき，レポートを作成すること。

補足

I.ラプラス変換

$t \geq 0$ において定義された $f(t)$ に対して，次式の定積分による変換をラプラス変換と呼ぶ。

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{-st}dt \tag{1-1}$$

このようなラプラス変換は，次式のように表す。

$$F(s) = L[f(t)] \tag{1-2}$$

ラプラス変換では，時間 t で表された時間領域の関数から， s で表された周波数領域の関数への変換を行っているとして，制御工学では多用される。

ラプラス変換の具体的な特徴や性質は，別途，勉強されたい。ここでは，制御において最低限必要な事項についてのみ触れる。

【線形性】

$t \geq 0$ における２つの関数 $f(t)$， $f(t)$ の線形和のラプラス変換は，次の様になる。

$$L[af(t) + bg(t)] = aL[f(t)] + bL[g(t)] \tag{1-3}$$

【導関数のラプラス変換】

$t \geq 0$ における関数 $f(t)$ の導関数 $\frac{d}{dt}f(t)$ のラプラス変換 $L\left[\frac{d}{dt}f(t)\right]$ は，関数 $f(t)$ のラプラス変換 $L[f(t)]$ を用いて次の様に表される。

$$L\left[\frac{d}{dt}f(t)\right] = sL[f(t)] - f(0) \tag{1-4}$$

【積分のラプラス変換】

$t \geq 0$ における関数 $f(t)$ の積分 $\int_0^t f(u)du$ のラプラス変換 $L\left[\int_0^t f(u)du\right]$ は，関数 $f(t)$ のラプラス変換 $L[f(t)]$ を用いて次の様に表される。

$$L\left[\int_0^t f(u)du\right] = \frac{1}{s}L[f(t)] \tag{1-5}$$

一方で， s 領域にある関数を t 領域に変換ことをラプラス逆変換と呼び，次式で定義される。

$$f(t) = L^{-1}\left[F(s)\right] = \frac{1}{2\pi j}\int_{c-j\infty}^{c+j\infty} F(s)e^{st}ds \tag{1-6}$$

制御工学では，物体の運動などの過渡応答を考える必要がある。そのため，時間領域（t 領域）では，微分および積分が重要となる。しかし，入出力が複数になるなど複雑な系では，微分方程式を解くこと自体が困難なことが多くなる。

一方で，ラプラス変換では，微分および積分が乗減算で表現された線形方程式としての解法が可能となる。そのため，制御工学では，ラプラス変換により微分方程式・積分方程式を，s 領域の線形方程式として考える。

ただし，本実験で扱うステップ応答のような応答を見る上では，時間領域で扱う必要があるため，ラプラス変換して演算した後，ラプラス逆変換により，時間領域に戻す。そのため，ラプラス変換だけでなく，ラプラス逆変換についても，理解しておく必要がある。

しかし，制御工学の概要を理解したいのであれば，ラプラス逆変換については，次の手順をとりあえず，理解しておけばよい。

Ⅱ.2 次遅れ系におけるステップ応答

伝達関数が,

$$G(s)=\frac{O(s)}{I(s)}=\frac{{\omega_n}^2}{s^2+\delta\omega_n s+{\omega_n}^2} \tag{2-1}$$

で与えられる系は，2 次遅れ系と呼ばれる。この 2 次遅れ系のステップ応答は，式(2-1)をラプラス逆変換することによって，次の様に,求まる。

$$O(t)=1-\frac{1}{\sqrt{1-\delta^2}}e^{-\delta\omega_n t}\sin\left(\omega_n\sqrt{1-\delta^2}\,t+\tan^{-1}\frac{\sqrt{1-\delta^2}}{-\delta}\right) \tag{2-2}$$

これをグラフ化すると，図 2-3 となる。

※ステップ応答

ある時間 t において，加速度∞で，絶対値 1 の値を入力したときの出力状態を見ること。

ステップ関数

$$y(t)=\begin{cases}0 & (t<t_1)\\ 1 & (t\geq t_1)\end{cases}$$

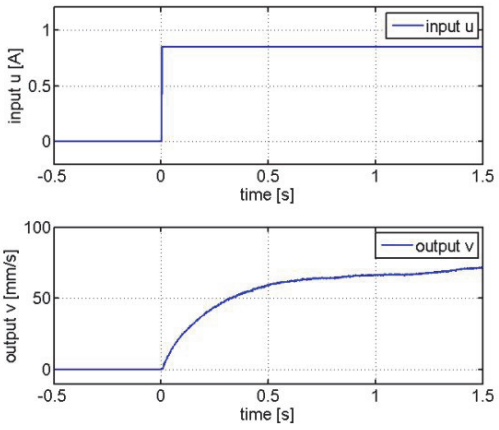


図 2-3　ステップ応答の概念

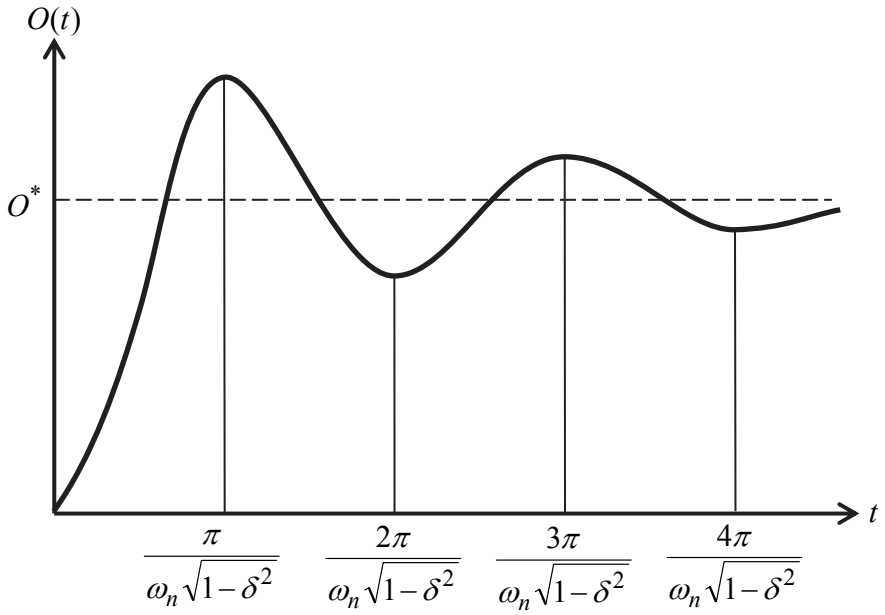


図 2-4　2 次遅れ系 $\frac{{\omega_n}^2}{s^2+\delta\omega_n s+{\omega_n}^2}$ のステップ応答

図 2-4 において，オーバーシュートによる最初のピークが来るまでの時間は，

$$T_O=\frac{\pi}{\omega_n\sqrt{1-\delta^2}}$$

で表される。その後は，一定周期で振動しながら，減衰していく。（ただし，この現象は，の場合であり，の場合には，オーバーシュートや振動しながらの減衰が現れない。）

なお，図 2-5 にステップ応答における代表的な応答例を示す。

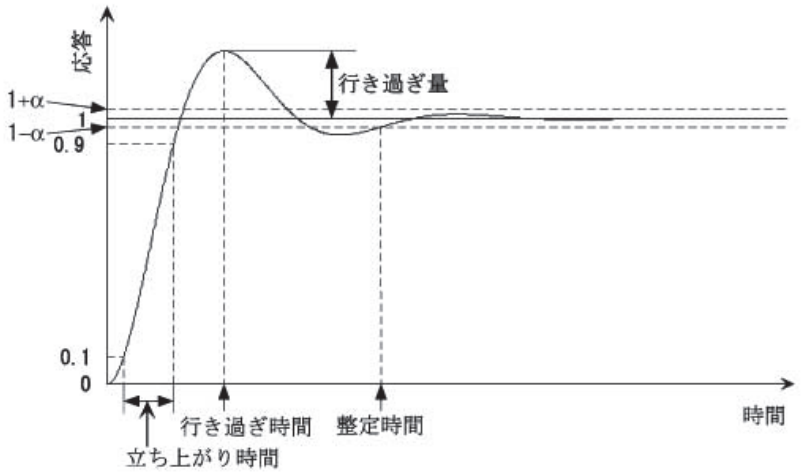


図 2-5　ステップ応答における応答例