コンピュータ理工学実験
コンピュータによる電子機器制御の基礎

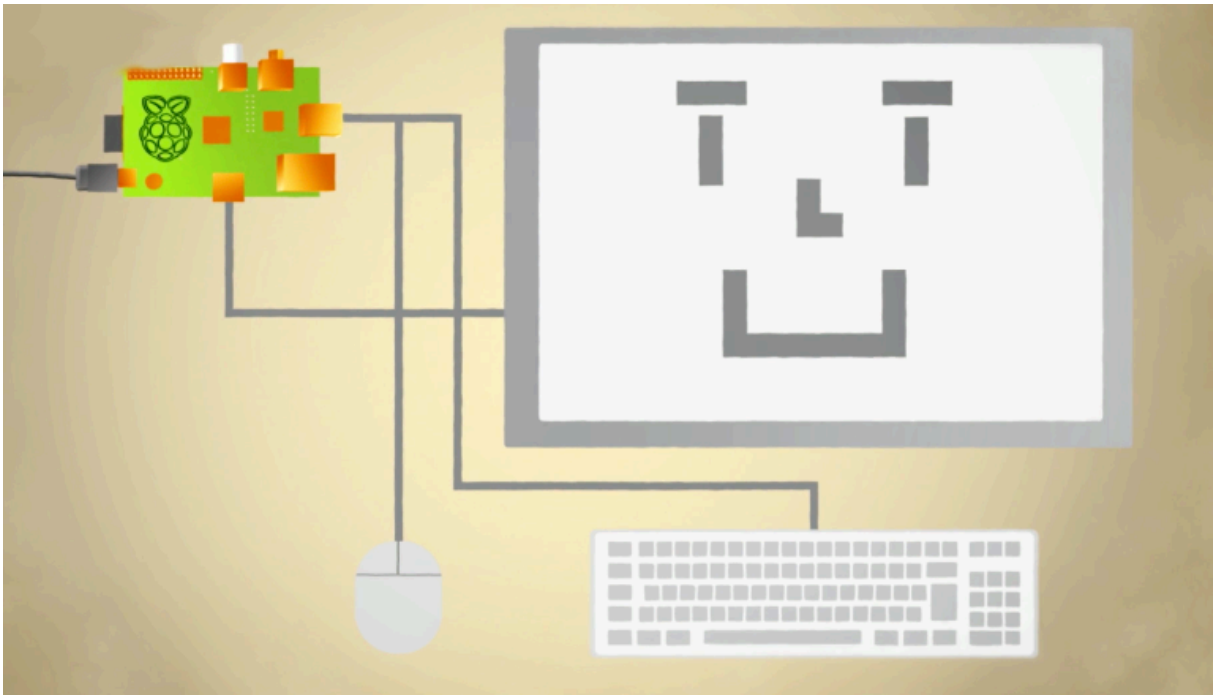
2017 年版 v3

概要

コンピュータの利用分野のひとつに，自動車などの電気機械装置を制御することや，そのような装置と利用者のインタフェースの役割を果たすものがある．このように，装置に組み込んで利用されるコンピュータシステムを組み込みシステムと呼ぶ．この実験では，組み込みシステムとしてのコンピュータの基本的な機能を学習する．

組み込みシステムにおけるコンピュータは，マイコンと呼ばれる CPU とメモリや IO 装置を備えた CPU 基板の形で利用される．電気機械装置の全体は，CPU 基板に装置固有の部品や回路・システム・機構部を接続することで構成される．

本実験では，CPU 基板として Raspberry Pi 2 を用いて 3 種類の実験を行う．第一の実験では，LED（発光ダイオード）とスイッチでできた簡単な回路を接続し，C 言語によるプログラムでこれらを制御する．第二の実験では，カメラモジュールを接続し，撮影した画像に処理を加えるプログラムを作る．第三の実験では，加速度・角速度センサと呼ばれるセンサ部品を接続し，その信号を用いて，物体の速度・位置と姿勢（角度）を求める．



目次

1章 組み込み機器に用いるコンピュータ

コンピュータはさまざまに利用されている．Google や Amazon のクラウドコンピューティングやスパコンのような超大型システムから，14 号館 1 階にあるサーバーマシンのような中型のシステム，そして，日常使っているパソコンやスマホが代表的なコンピュータである．実際にはこれらだけではなく，ほとんどの電気機械装置にはコンピュータが組み込まれている．原子力発電所・飛行機・自動車や交通制御システム・自動改札・銀行の ATM・医療機器・テレビ・カメラ・プリンタ・冷蔵庫・炊飯器など，数え上げればきりがない．

装置に組み込まれるコンピュータは専門化した機能を持ち，動作の信頼性が高い．大量に生産するものは，消費電力が少なく，小型で安価である．このようなものを組み込み用コンピュータとよぶ．図 1.1 に，入手が容易な組み込み用コンピュータの例を示す．

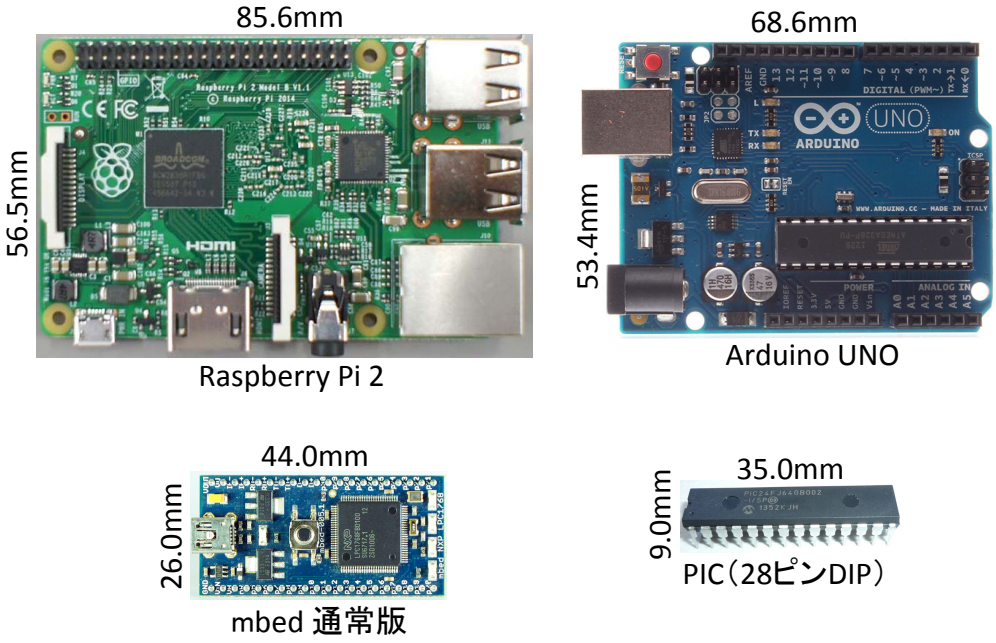


図 1.1 組み込み用コンピュータの例

Raspberry Pi はカードサイズの CPU ボードに，周辺機器を接続できる入出力機能を加えたものである．基本的には Linux パソコンであり，４つの中では格段に処理性能が高い．プログラム言語は Python・C・Java のいずれかを用いることが多い．ソフト開発は比較的容易であるが，Linux で動作するため厳密なリアルタム処理には向かない．Arduino はオープンソース／オープンハードを特徴とし，組み込み用コンピュータとして古くから実績がある．OS はなく，C 言語風プログラム言語を使う．mbed は arduino を小型にしたような装置である．PIC は，比較的簡単な命令語を持つ CPU コアに，ROM・RAM，タイマー・割り込みコントローラ，周辺機器とのインタフェース回路を一体化したマイコンチップである．非常にたくさんのバリエーションを備えている．プログラムは C 言語かアセンブラで記述する．他の３つが主に試作用の部品であるのに対して，実製品にも広く利用される．

2章 Raspberry Pi 入門

2.1 Raspberry Pi の概要

Raspberry Pi はカードサイズの CPU 基板である．Linux 系 OS が動作し，ディスプレイ，キーボード，マウス，ネットワークを接続することで一台のパソコンとして利用することができる．いくつかのモデルがあり，この実験では Raspberry Pi 2 Model B（以下，本文では RasPi2 と略記する）を利用する．表 2.1 に，RasPi2 の主な仕様を示す．

表 2.1 RasPi2 の主な仕様

CPU，メモリ	ARM Cortex-A7 4コア/900MHz，1GB
主なインタフェース	USB×4，HDMI，イーサネット
主な入出力ポート	40ピンのピンヘッダを GPIOなどに利用可能
ストレージ	micro SDカード
電源，寸法	5V・900mA，85.6mm×56.6mm

RasPi2 に搭載されている主な部品の名称と機能を，図 2.1 に示す．microSD カード（以下，本文中では SD カードと記す）はシステムプログラム（OS）とユーザプログラムやデータを格納する外部メモリとして利用する．CPU は 4 コアの ARM Cortex-A7 である．ボード中央にあるチップは，CPU を始めとする多くの機能を一つのチップに集積した SoC（System on Chip）である．ディスプレイ，キーボード，マウス，イーサーネットは，それぞれ，HDMI コネクタ，USB 端子，イーサーネット端子に接続する．電源はマイクロ USB の電源コネクタを介して供給する．電源スイッチは無く，コネクタに電源プラグを接続することでスイッチが入る．電源を切る場合にはあらかじめシステムをシャットダウンし，動作が止まったことを確認（緑の LED が点滅しなくなる）して電源プラグを抜く．また，40 ピンのピンヘッダやコネクタを介して外部の機器・装置・センサ・モーターなどを接続できる．

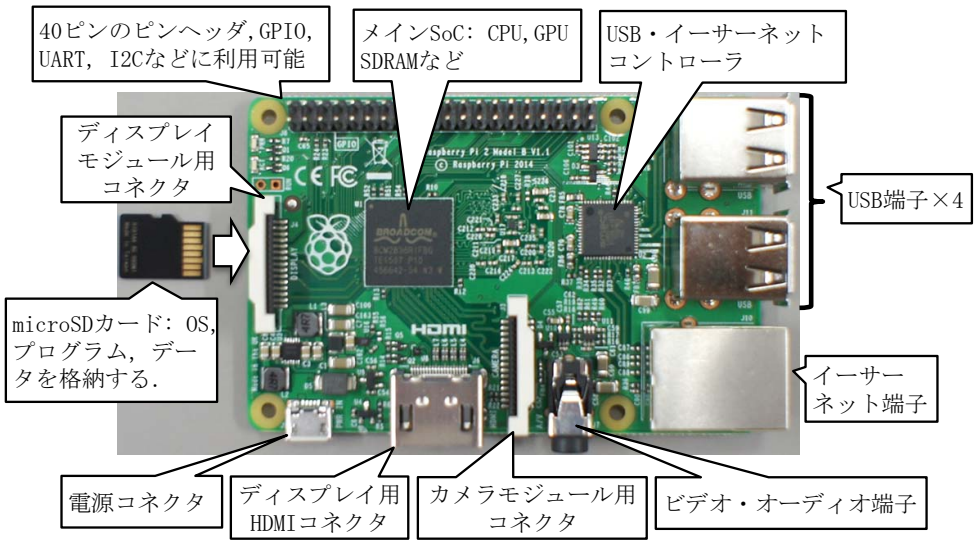


図 2.1　RasPi2 と各部の説明

2.2 OS のインストール（本実験ではインストール済み）

一般ユーザは，Raspberry Pi の機能を OS(Operating System，オペレーティングシステム)を使って利用する．Raspberry Pi には複数の Linux 系 OS が開発されており，この実験では最も代表的な Raspbian を用いる．

SD カードに Raspbian をインストールする方法に，起動ディスクイメージを直接 SD カードに書き込む方法と，NOOBS（New Out Of the Box Software）を用いる方法がある．付録 1 に NOOBS を用いる方法を説明する．Raspbian のインストールに続いて幾つかの初期設定を行い，日本語環境を導入する．付録 3 に，SD カードイメージを Mac に保存する方法と，Mac に保存された SD カードイメージを書き込む方法を示す．

なお，実験では，必要なソフトウエアが導入済みの SD カードを提供するので，これらの手順は必要ない．

2.3 接続・起動・利用・終了

Raspbian がインストール済みの SD カードを RasPi2 に装着する．USB マウス，USB キーボード，イーサーネット，ディスプレイを図 1.2 のように接続する（本実験ではイーサーネットに接続する必要はない）．この状態でマイクロ USB コネクタに電源を接続するとシステムが起動する．

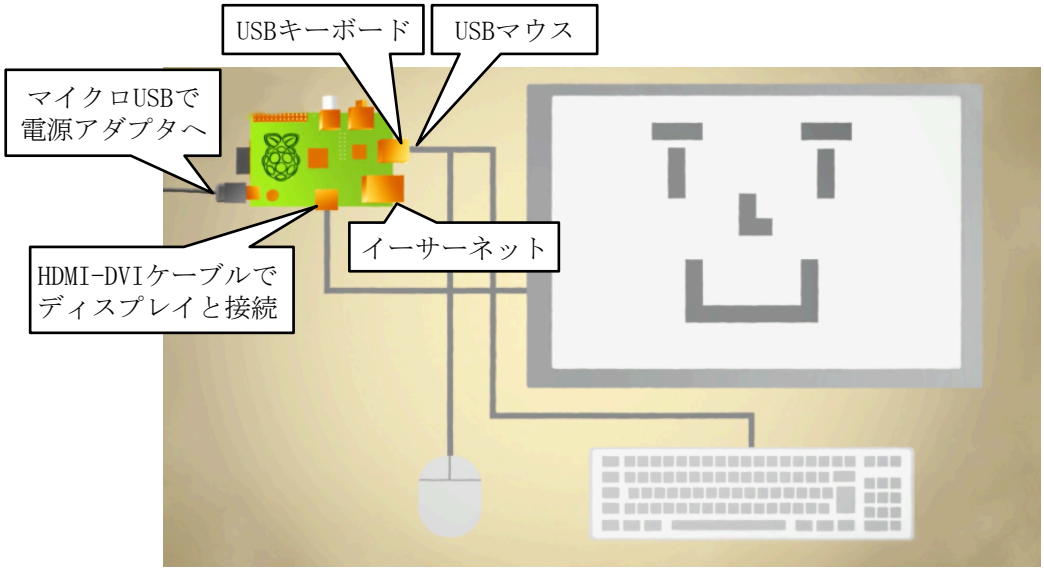


図 2.2　RasPi2 の接続

システムが起動すると，最終的に，図 2.3 のデスクトップ画面が表示される．デスクトップ環境の操作は Mac や Windows と似たようなものである．図 2.3 に初期状態で表示されている主なアイコンの意味を示す．この中で，実験に利用するものは，プルダウンメニューのアクセサリにある Emacs，LeafPad と呼ばれる Text Editor，ターミナル，ファイルマネージャなどである．

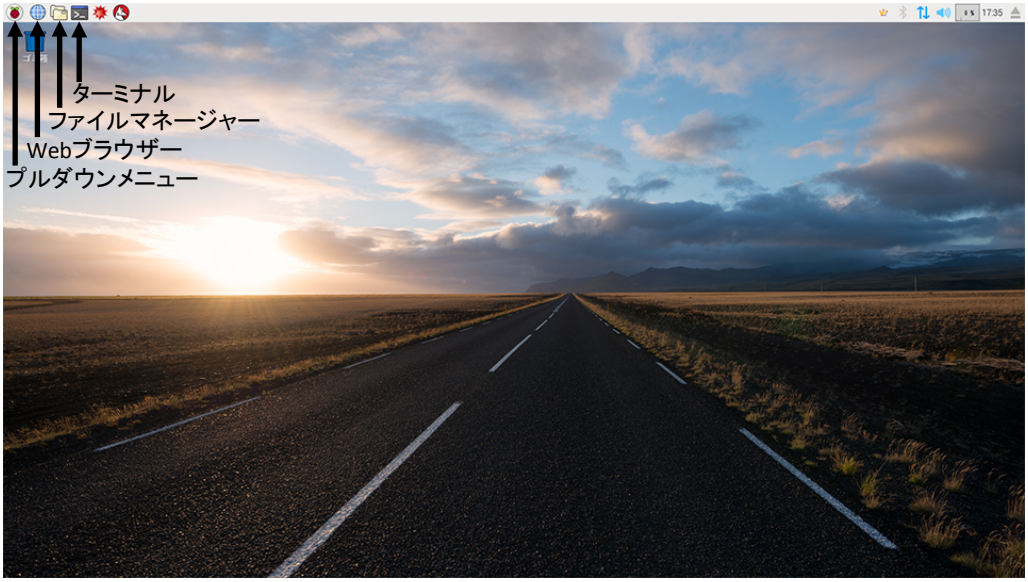


図 2.3　デスクトップ画面

RasPi2 を終了するには，Menu からシャットダウンを選択するか，ターミナルを起動し

```
$ sudo shutdown –h now
```

あるいは

```
$ sudo halt
```

と入力する．ここで，sudo は root（ルート，スーパーユーザー，管理者）としてコマンドを実行させることを意味している．一般ユーザとして実行できないコマンドであっても，その前に sudo を入れることで，実行できるようになる．正しくシャトダウンするとディスプレイの表示が消え，最後にボード上の緑色 LED が消える．この段階で電源プラグを抜く．緑色 LED は SD カードへのアクセスが発生していることを示す．これが点灯・点滅している時に電源を切断すると，SD カードの内容が破壊される可能性が高い．

RasPi2 で利用できるテキストエディタの代表は nano と LeafPad である．また，emacs もインストールしてある．Vi も利用できる．nano はターミナルから利用するエディタ，LeafPad は GUI で操作するエディタである．例えば，nano を使ってシステムファイルを編集する場合，ターミナルから

```
$ sudo nano  システムファイル名
```

のように入力して nano を起動する．一般ユーザは，システムファイルを修正できないため，このように sudo をつける必要がある．図 2.4 に nano の画面を示す．画面下部にコマンドが表示されている（^X はコントロールキーを押しながら X を押すことを意味する）．LeafPad は Menu からアクセサリ→Text Editor と選択して起動する．なお，emacs はインストールしたままのデフォルト状態であるから，プログラミング演習で利用している環境とは，操作感が少し異なっている．この実験では，自分の使いやすいエディタを使えばよい．（特にこだわりがなければ emacs を使ってください）

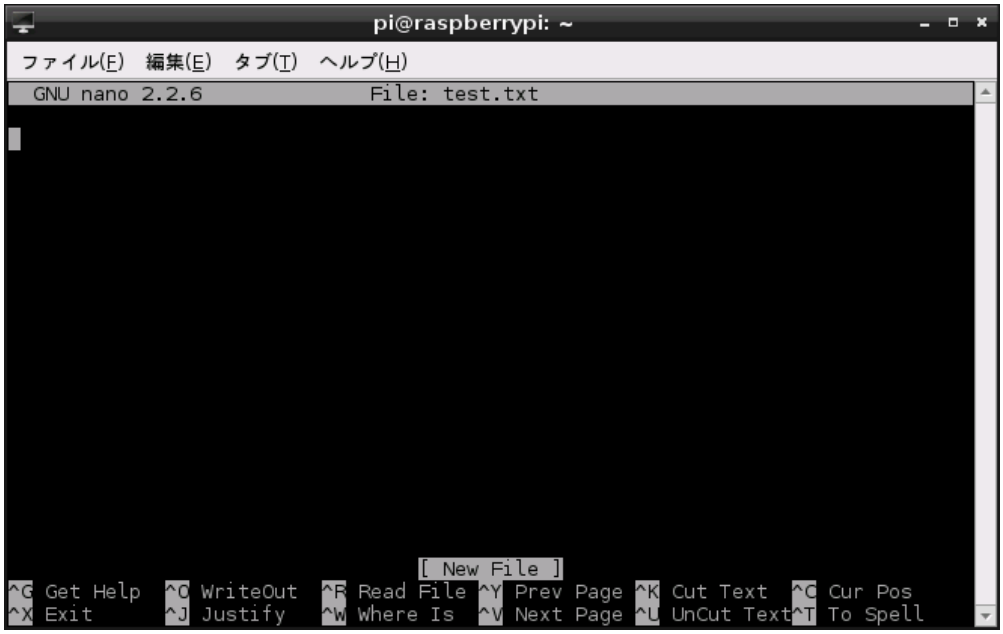


図 2.4　テキストエディタ nano の画面

2.4 Linux 入門

SD カードにインストールした Raspbian は，Linux ディストリビューションの一つである Debian を Raspberry Pi 用に最適化した OS である．Mac の OS X とは，共に UNIX 系 OS であるという意味で親戚と言える．実験で利用する RasPi2 の機能の多くは，ターミナルから Linux コマンドを入力することで利用する．（Linux コマンドは Mac のターミナルから入力するコマンドとほぼ同じである）

2.4.1 Linux コマンドの例

デスクトップ環境でターミナルを起動すると，ターミナル画面に

pi@raspberrypi ~ $

と表示される．これはコマンド入力待ちであることを示す．pi はユーザ名，raspberrypi はコンピュータ名（ホストネーム），~はホームディレクトリ（/home/pi）を示す．Linux のファイルシステムは，ディレクトリがツリー状に構成されている．それぞれのディレクトリにファイルと下位のディレクトリが保存される．代表的なコマンドの例を次に示す．

$ cd ..	カレントディレクトリを一段階上がる
$ cd /	カレントディレクトリを/（ルート）にする
$ ls	カレントディレクトリのファイルの一覧を表示する
$ ls -l	ファイルの一覧を詳細に表示する
$ ls -la	ドットファイルを含めてファイルの一覧を詳細に表示する
$ touch foo	空ファイル foo を生成する．foo が存在する場合，作成日時を更新する．
$ mv foo baz	foo の名前を baz に変更する

$ cp foo baz	foo と同じ内容のファイル baz を作る
$ rm baz	ファイル baz を削除する

空のディレクトリを削除するには rmdir，中にファイルが入っているディレクトリを削除するには rm –r を用いる．rm コマンドにオプション-r をつけると「ディレクトリの中身を再帰的に削除せよ」という意味になる．

$ man curl	curl コマンドのマニュアルを表示する
$ rm --help	rm コマンドのヘルプを表示する
$ mkdir myDir	新しいディレクトリ myDir を作る
$ grep Puzzle */*	カレントディレクトリの下にある全ファイルで文字 Puzzle を検索する
$ sudo find / -name “stdio.h”	stdio.h という名前のファイルを，/以下の全ての場所から探す

2.4.2 パイプとリダイレクション

Linux（UNIX）の機能にパイプとリダイレクションがある．パイプは２つのプログラムを連携して機能させるしくみで，第一のプログラムの標準出力（stdout）を第二のプログラムの標準入力（stdin）に接続する機能である．例えば，

```
$ ls -la | less
```

とすることで, ls -la の出力を less コマンドの形式で表示する．ここで‘|’がパイプを表す．

リダイレクションはプログラムの標準出力（stdout）をファイルに向ける機能である．

```
$ ls > list.txt
```

とすることで，ls の出力を list.txt というテキストファイルに保存することができる．

```
$ cat wibble.txt wobble.txt > wobble.txt
```

とすることで，２つのファイル wibble.txt と wobble.txt をつないだファイル wobble.txt を作ることができる．‘>’がリダイレクションの記号である．

2.4.3 プロセス

Linux が管理するプログラムの単位をプロセスとよぶ．Linux は，常時，数十個のプロセスが動作している．ps コマンドは実行中のプロセスの状態を表示する．

```
$ ps -aux
```

とすることで, 実行中の全てのプロセスの状態を表示する. プログラムを強制終了するには,

```
$ ps –aux | grep プログラム名
```

のようにして，プログラムのプロセス ID を調べ,

```
$ kill プロセス ID
```

で強制終了させる．自分には権限がないプロセスを終了させる場合

```
$ sudo kill プロセス ID
```

とする必要がある．ただし，権限がないプロセスの終了は慎重に行う事．

2.4.4 パーミッション

Linux では，一般ユーザに権限がない操作であっても，root ユーザとしてコマンドを実行することができる．root としてコマンドを実行する場合，通常のコマンドの前に sudo をつけて実行させる．しかし，root としてコマンドを実行することはシステムに大きなダメージを引き起こすこともあるので，特に注意すること．

すべてのファイルとディレクトリは，一人のユーザと一つのグループに属している．chown と chgrp のコマンドを使うと，所有者とグループを変更することができる．

`$ sudo chown pi garply.txt`　　garply.txt の所有者を pi にする

`$ sudo chgrp staff plugh.txt`　　plugh.txt のグループを staff にする

各ファイルとディレクトリは読む（r），書く（w），実行する（x）という 3 種類のパーミッション（許可）が設定されている．その設定状態は，ls -l コマンドでファイルの詳細情報を表示することで，図 2.5 のように確認することができる．パーミッションは chmod コマンドで変更することができる．chmod コマンドは表 2.2 の略号を使い，

`$ chmod u+rwx, g-rwx, o-rwx wibble.txt`　　所有者のみが rwx 可能とする

`$ chmod g+wx wobble.txt`　　グループに wx の権利を追加する

`$ chmod -rwx, +r wobble.txt`　　全ユーザに r だけを許可する

のようにパーミッションを設定する．あるいは，rwx を 3 桁の 8 進数に見立てて，

`$ chmod 755 wobble.txt` のように設定することも可能である．これによって所有者は rwx 全て可能，その他のものは r と x が可能となる．

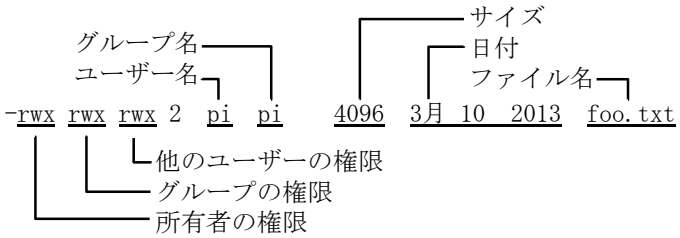


図 2.5　ファイルのパーミッション情報

実験中に，実行できるべきファイルが実行できない，という状況になったときは，

`$ chmod 755 ファイル名`

のようにして，実行権限を与えると解決する場合が多い．

表 2.2　chmod コマンドで使われる略号

u	所有者
g	グループ
o	他のユーザー
a	全員
r	読み込み権限
w	書き込み権限
x	実行権限
+	権限の追加
-	権限の削除

2.4.5 ソフトウエアのインストールとアップデート

ネットワークに接続した Raspberry Pi の IP アドレスを調べる場合，ifconfig コマンドを用いる．

```
$ ifconfig
```

他のコンピュータと通信可能かを調べる場合には ping コマンドを用いる

```
$ ping yahoo.com
```

ネットワークを介してソフトウエアをインストールするには，多くの場合，APT とよばれる Debian 用のパッケージ管理システムを用いる．例えば，次のコマンドで emacs エディタがインストールできる（実際にインストールする必要はない）．

```
$ sudo apt-get install emacs
```

APT が管理するソフトウエアパッケージは頻繁に更新される．そのパッケージリストを更新し，インストールしたソフトウエアをアップデート・アップグレードするために，

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

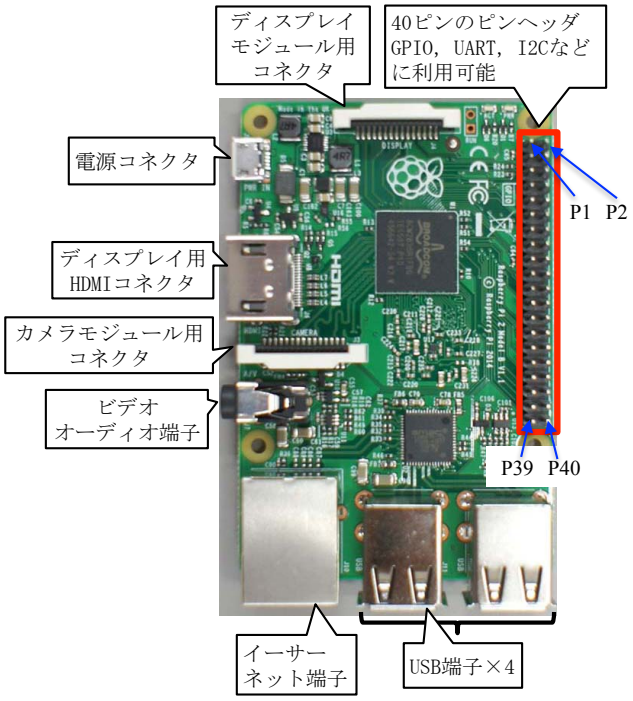
を行う（この実験では，これらを行う必要はない）．

3 章 実験 1：Raspberry Pi を使ったスイッチと LED の制御

3.1 Raspberry Pi と周辺装置・周辺回路の接続

RasPi2 が普通の Linux パソコンと異なっている点のひとつは，周辺装置やセンサなどの電子部品との接続が容易なことである．図 3.1 に RasPi2 の外部接続端子と 40 ピンの端子列（ピンヘッダ）のピンアサインを示す．

この中で，40 ピンの端子列は図の右表のようにさまざまな役割で利用することができる．GND（電気回路を接続するときに基準となる電圧で 0V），+3.3V，+5V はそれぞれの電圧が供給される．GPIO2 など GPIO で始まるものはデジタル信号（0 か 1）の入出力に利用する端子である．I2C1 SDA など独特な名前の端子は，ある規格に基づいてデジタル信号を入出力する端子である．これらの名前から，Raspberry Pi が I2C，SPI，UART などの規格をサポートしていることがわかる．したがって，これらの規格を持つ装置と容易に接続することができる．なお，“GPIO2, I2C1 SDA”などのように 2 つの名称が併記されている端子は，プログラムによってそれらの機能を切り替えることができる．



+3.3V	P1	P2	+5V
GPIO2, I2C1 SDA	P3	P4	+5V
GPIO3, I2C1 SCL	P5	P6	GND
GPIO4, GPCLK0	P7	P8	GPIO14, UART TxD
GND	P9	P10	GPIO15, UART RxD
GPIO17	P11	P12	GPIO18, PCM_CLK
GPIO27, PCM_DOUT	P13	P14	GND
GPIO22	P15	P16	GPIO23
+3.3V	P17	P18	GPIO24
GPIO10, SPI MOSI	P19	P20	GND
GPIO9, SPI MISO	P21	P22	GPIO25
GPIO11, SPI SCLK	P23	P24	GPIO8, SPI CE0
GND	P25	P26	GPIO7 SPI CE1
ID_SD	P27	P28	ID_SC
GPIO5	P29	P30	GND
GPIO6	P31	P32	GPIO12
GPIO13	P33	P34	GND
GPIO19	P35	P36	GPIO16
GPIO26（入力で使う）	P37	P38	GPIO20
GND	P39	P40	GPIO21（出力で使う）

ピンヘッダのピンアサイン

図 3.1 RasPi2 の外部接続端子とピンヘッダのピンアサイン

3.1 GPIO 端子の設定と入出力

GPIO（General Purpose Input Output，汎用入出力）はさまざまな用途に利用することができる入出力端子という意味である．Raspberry Pi のプログラムで GPIO 端子を入力に設定したあとで，その端子に 0V の電圧を加えれば，その状態を値 0 としてプログラムに読み込む．加える電圧が 3.3V であれば値 1 として読み込む．（Raspberry Pi の GPIO 端子に加え

ることができる電圧は 0V～3.3V の範囲である．5V の電圧を加えると回路が壊れてしまうので，注意する事.）GPIO 端子を出力に設定し，プログラムがその端子を値 0 に設定すると，実際の出力電圧は 0V になる．値 1 を設定すると，出力電圧は 3.3V になる．

それでは，実際に GPIO 端子をプログラムしてみる．この実験では，C 言語のプログラムから WiringPi というライブラリを使って，RasPi2 の GPIO を制御する．GPIO21（P40）を出力に設定して値 1 を出力し，GPIO26（P37）を入力に設定してその値を読み込み，ディスプレイに表示するプログラムの例を図 3.2 に示す．これを

```
$ gcc –o sample sample.c –lwiringPi
```

でコンパイルする．-lwiringPi はコンパイラに対するオプションで，ライブラリ wiringPi を利用することを意味する．実行ファイル sample が出力されれば，

```
$ sudo ./sample
```

で実行させる．wiringPi を使うプログラムは sudo で実行する必要がある．プログラムの出力は“GPIO26: ?”（?は 0 か 1）となり，GPIO26 の端子の電圧値がそのように解釈されたことがわかる．この段階で，GPIO26 の端子になにも接続していなければこの値にあまり意味はない．GPIO26（ピンヘッダの 37 番端子）と GPIO21 （ピンヘッダの 40 番端子）の端子を接続しておれば，“GPIO26: 0”となるはずである．

```
#include <stdio.h>
#include <wiringPi.h>

#define GPIO21 21
#define GPIO26 26

int main() {
        //wiringPi の初期化
        if (wiringPiSetupGpio() == -1) {
                printf("error: wiringPiSetupGpio()\n");
                return 1;
        }

        //GPIO21 を出力に、GPIO26 を入力に設定する
        pinMode(GPIO21, OUTPUT);
        pinMode(GPIO26, INPUT);

        //GPIO21 に 1 を出力し、GPIO26 の信号を読み取り表示する。
        digitalWrite(GPIO21, 0);
        int sw = digitalRead(GPIO26);
        printf("GPIO26: %d\n", sw);
}
```

図 3.2 sample.c，GPIO 端子を設定するプログラムの例

プログラムの内容を確認する．関数 pinMode()で端子を入力か出力に設定する．pinMode(GPIO21,OUTPUT)は GPIO21，すなわち 40 番の端子を出力に設定する．pinMode(GPIO26,INPUT)は GPIO26，すなわち 37 番の端子を入力に設定する．

関数 digitalWrite()で出力端子に信号値を設定する． digitalWrite(GPIO21,0)にすると 40 番の端子電圧は 0V になる．digitalWrite(GPIO21,1)にすると 40 番の端子電圧は 3.3V になる．関数 digitalRead()で入力端子から信号値を読みとる．37 番の端子に 0V を加えた状態で digitalRead(GPIO26)を実行すると，その端子の値を 0 として読み取る．37 番の端子に 3.3V を加えた状態で digitalRead(GPIO26)を実行すると，その端子の値を 1 として読み取る．

pinMode()，digitalWrite()，digitalRead()は WiringPi に存在する関数である．このようにあらかじめ準備してある関数を自分のプログラムで利用すること“リンク”する，という．

3.2 電気信号を接続する

この実験では，ブレッドボードとジャンパ線を使って簡単な電気回路を作る．ブレッドボードは多数の穴が配置され，穴が内部で規則的に接続されている．図 3.3 左の製品では，左右両端の縦 4 列は列ごとに全ての穴がつながっている．それ以外の穴は，横方向に 5 つの穴がつながっている．ジャンパ線は，内部接続されていない穴同士を接続するために用いる配線である．赤いジャンパ線を図 3.3 左のように接続すると，赤色の点線で示した位置にある穴が全て導通する．

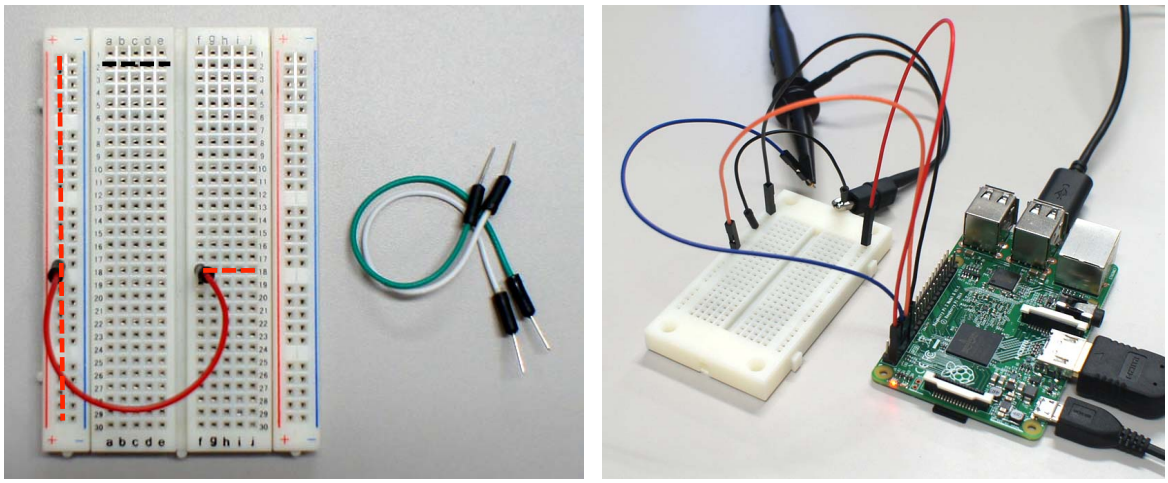


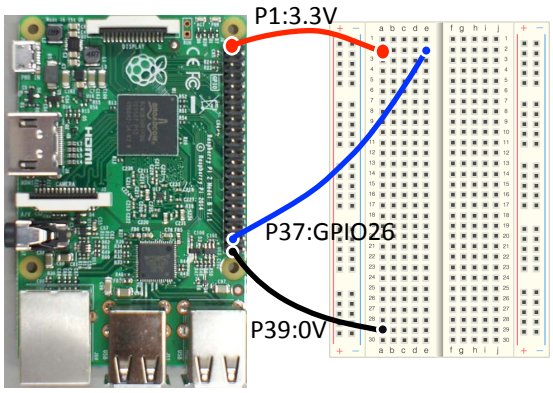
図 3.3 左：ブレッドボードとジャンパ線，右：接続の例

ここから先の実験では実際に電気回路を作る．間違えて GND と 3.3V や 5V を接続したり，GPIO 端子に 5V を接続すると，RasPi2 が壊れる．十分に注意して実験を進めること．このような致命的な間違いでなくとも，出力に設定した端子同士を接続することや，出力端子に GND や 3.3V あるいは 5V を接続すると，その端子が壊れる可能性がある．

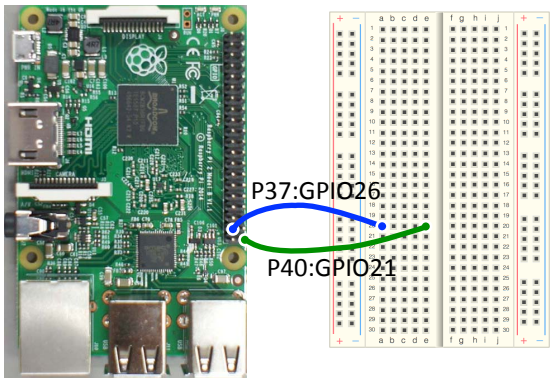
ブレッドボードとジャンパ線を使って RasPi2 のピンヘッダの端子を接続し，プログラム sample.c の動作を実際に確認しよう．これを行う回路の実体配線図の例を図 3.4 に示す．図

3.4 左の回路は，GPIO26 の端子に 3.3V を加えた状態である．この状態で sample.c を実行すると 1 が出力されるはずである．GPIO26 の端子に 0V を加えるように変更し，sample.c を実行するとプログラムは“GPIO26: 0”と出力するはずである．

図 3.4 右の回路は GPIO21 を GPIO26 に接続している．この状態で sample.c を実行すると 0 が出力される．GPIO21 に 1 を出力するように変更し，sample.c を実行すると 1 が出力される．（図 3.3 の右側の写真や図 3.4 の実体配線図は，回路の接続状態を説明するために便宜的に用いているのであり，正式な回路図ではない．レポートで回路を説明するときには，正しい回路図を使うこと．）



GPIO26に対応する入力端子P37に，3.3Vの電圧を加えたり0Vの電圧を加えるための回路の例．



GPIO21から出力される信号をGPIO26で読み取る回路の例．

図 3.4 GPIO21, GPIO26 の端子の動作を確認する実体配線図の例

3.3 GPIO 入力端子にスイッチを接続する

前節の実験では入力端子に電圧を加え，0V と 3.3V の違いを，論理値の 0/1 で検出した．この節では，これをスイッチに置き換え，スイッチの ON/OFF を 0/1 で検出する．

図 3.5 にタクトスイッチを示す．このスイッチは 4 本の端子があり，図の上側の 2 本・下側の 2 本は内部でつながっている．スイッチを押すと上下の端子が導通する．図 3.5 右に，スイッチの ON/OFF によって，「スイッチの出力」と書かれた部分の電圧を 0 V と 3.3V に切り替える回路の例を示す．スイッチが OFF の時，スイッチ下部は抵抗を介して GND に接地されるので，そこの電圧は 0V になる．スイッチを ON するとスイッチ下部の電圧は 3.3V になる．図 3.5 の回路を作り，スイッチの ON/OFF を sample.c のプログラムを使って GPIO26 から読み取り，その動作を確認せよ．

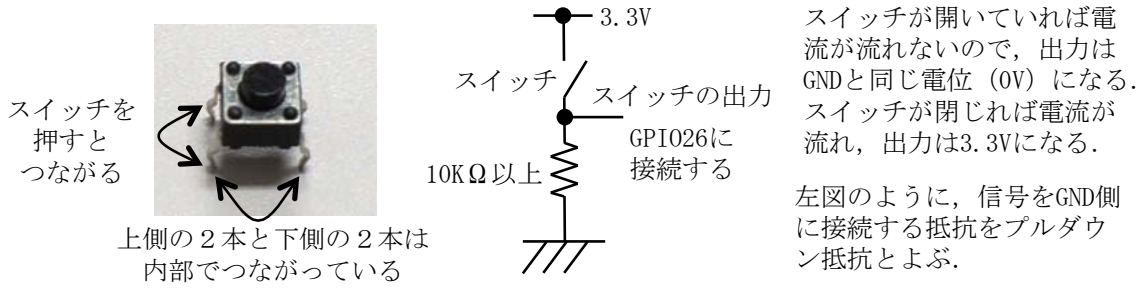


図 3.5 タクトスイッチと，その ON/OFF を 0V/3.3V に変換する回路の例

3.4 GPIO 出力端子に LED を接続する

LED（Light Emitting Diode，発光ダイオード）は電流を流すと発光する半導体である．図 3.6 に LED の外観と，GPIO 出力端子で LED の発光を制御する回路の例を示す． LED はアノード，カソードとよぶ２つの端子を持ち，アノード（リードが長いほう）からカソード（リードが短いほう）に適量の電流を流すことで発光する．例えば，図 3.6 右のように配線し，GPIO21 の電圧を０（0V）にすると LED が消灯し，１（3.3V）にすると点灯する．

まず，GPIO21 の出力を，周期的に０と１（0V と 3.3V）に切り替えることで，LED を点滅させてみよう．点滅は，点灯と消灯を比較的ゆっくり繰り返すことであり，そのためには，一定の時間を待つ必要がある．プログラムで一定時間待つには， wiringPi にマイクロ秒単位の遅延を入れる関数 delayMicroseconds(マイクロ秒)があるので，delayMicroseconds(1000000)のように使うとよい（これで１秒の遅延になる）．これらの回路の実体配線図の例を図 3.7 に示す．（この実験で GPIO21 を直接アノードに接続する学生を見かける（抵抗を使わないということ）．その場合，電流が多大に流れるので LED は明るく光るが，LED や GPIO 出力端子が劣化する．必ず，図 3.6 のように配線すること）

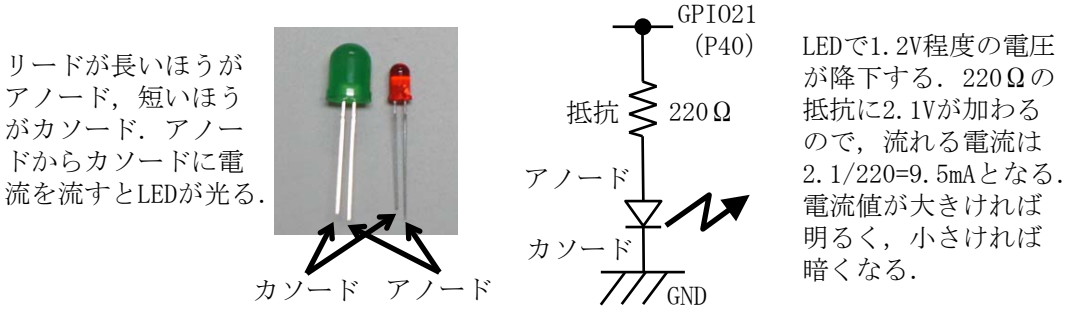
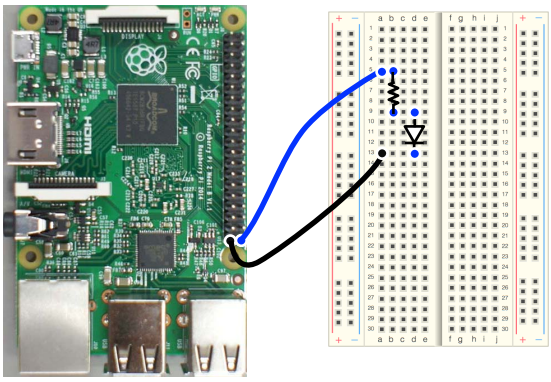


図 3.6 LED を発光させる回路の例



GPIO21に対応する出力端子P40でLEDの点滅を制御する回路の例.

図 3.7 LED を発光させる回路の実体配線図の例

led-control.c として，delayMicroseconds()を使って短い時間間隔で LED をオン・オフさせるプログラムを用意した．このプログラムを参考に，点滅の時間間隔を短くして，どの程度の周波数であれば点滅を目で認識できるか報告せよ．また，LED の点滅が目視確認できない周波数（例えば 100Hz や 1kHz）に設定し，周期に対するオン時間の比率を変えると，LED の明るさがどのように変化するかを調べよ．このとき，周期を一定にするため，オン時間+オフ時間は一定に保つこと．

LED に加える電圧をオン／オフし，そのオン／オフの時間を変化させることで明るさを制御する技術を PWM（Pulse Width Modulation）と呼ぶ．図 3.8 に PWM で LED を制御するタイミングを示す．LED は周期的に点滅している．その周期が長ければ，人間の目に点滅していることが見える．周期が早くなれば点滅の見分けがつきにくくなる．周期が十分に早ければ，点滅していることが全くわからない．このとき通常の点灯のように見え，その LED の明るさは周期に対する onTime の比率（デューティ比）によって決まる．

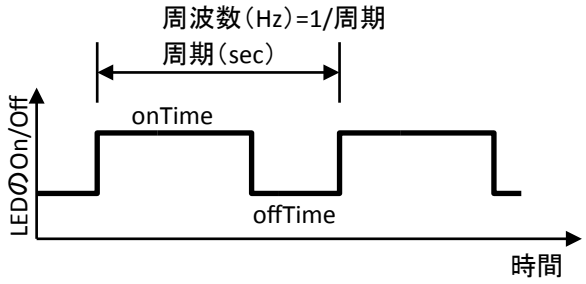


図 3.8 PWM の説明図

ここでは，PWM で明るさを制御したが，モーターに加える電圧を PWM 制御し，回転速度を調節することなどにも使う．PWM 制御について調査し，技術内容，メリット・デメリットなどを報告せよ．

3.5 実験１の作成課題

スイッチを押すごとに LED の明るさが３段階に変化する装置を作れ．プログラムを開始した時点では，LED は消灯している．スイッチを押すと，暗く点灯する．スイッチを離してもその状態が維持される．次にスイッチを押すと中間の明るさで点灯する．スイッチを離してもその状態が維持される．次にスイッチを押すと明るく点灯する．スイッチを離してもその状態が維持される．次にスイッチを押すと LED は消灯する．スイッチを離してもその状態が維持される．以降，同じ動作を繰り返す．

いくつか，注意するべきことがある．LED の明るさは PWM を使って調整する．スイッチを離しても LED の明るさを維持するためには，スイッチが押された瞬間を検出する必要がある．スイッチを押した瞬間にチャタリングと呼ばれる現象が起きる可能性がある．これは，１回のオン／オフ動作であるにもかかわらず，接点での接触が不安定になり，プログラムが数回のオン／オフを検出してしまう現象である．LED の明るさを３段階（消灯を合わせると４段階）に制御するためには，その状態を記憶する変数が必要である．これらの注意事項をどのように解決したかを報告すること．

仕様どおりに動作しない場合，その原因・解決手段を十分に考えて報告すること．

以上で実験１を終了する．

3.6 実験１のレポートについてのアドバイス

タイトル：「Raspberry Pi を使ったスイッチと LED の制御」

概要：最初に 300 文字程度で概要（この実験の成果を短くまとめたもの）を書くこと．

本文：次のような章立てで記述すること．具体的な章・節のタイトルは自分で考えること．また，自分の考えがあれば，章立てを変更しても良い．この資料で，青字で示した部分に対しては，レポート内に解答を記述すること．

１章	この実験の目的を書く．
２章	実験の前提になっていることについて書く．Raspberry Pi の GPIO を外部の回路に接続する方法など．
３章	プログラムを作成する前に行った検討内容について書く．スイッチの状態を入力する方法や，LED の点滅・明るさを制御する方法を書く．
４章	作成したプログラムについて書く．どのように問題を解決したか．解決できなかった問題はなにか，などを書く．
５章	まとめる

今回のレポートの１章や２章の最初に，次のようなことについて書くと良い．① 組み込み装置とはなんであるか．② 組み込み装置に使うコンピュータはどのような性質を持っているか．③ Raspberry Pi に使われる ARM という CPU は，組み込み装置や IoT（Internet of

Things）に適していると言われるが，その理由はなぜか．④ マイコンを使って周辺装置を制御するというのは，具体的にはどのような操作をすることか．

本文中で指示したことに従うこと．「CSE 実験レポートの書き方」を参考にして，わかりやすい日本語で書くこと．適切に図表を挿入すること．

4 章 実験 2：Raspberry Pi と OpenCV を使った画像処理

この実験ではカメラモジュールを RasPi2 に接続し画像を入力する．そして，カメラから入力した画像を，OpenCV を使って処理する．その後，OpenCV の関数で行った画像処理を自分のプログラムで実装する．

4.1 カメラモジュールの装着

カメラモジュールは電子部品としてのカメラのことである．画像センサ・レンズ・抵抗やコンデンサなどの部品を実装した小さい基板とケーブルなどからできている．この実験で使うカメラモジュールは，5M ピクセル（500 万画素）と 8M ピクセル（800 万画素）の CMOS 画像センサを使っている（Raspberry Pi カメラモジュールの V1 と V2）．これを，15 ピンのフレキシブルケーブルで，RasPi2 ボードの CSI（Camera Serial Interface）コネクタに装着する．CSI コネクタのカバー（白い部分）を引き上げ，フラットケーブルの金属端子が図 4.1 で手前側に来るように挿入する．ケーブルを奥まで差し込んだ状態でカバーを押し込む．
RasPi2 にカメラモジュールを接続する場合には，必ず，RasPi2 の電源を切っておくこと．

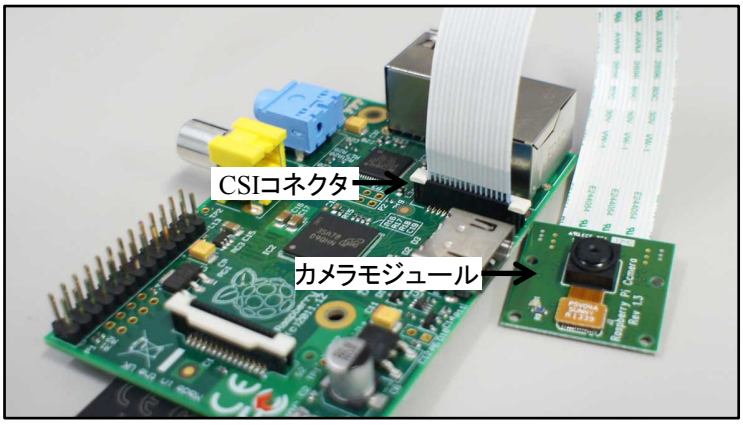


図 4.1 カメラモジュールと基板への装着

4.2 コマンドツールによる静止画と動画の撮影

Raspberry Pi には静止画を撮影するコマンドツール raspistill と，動画を撮影するコマンドツール raspivid が用意されている．これらを使ってカメラモジュールの動作を確認しよう．

```
$ raspistill –o image.jpg
```

と入力すると，ディスプレイに 5 秒間プレビュー映像を表示した後，最終時点の静止画をファイル名 image.jpg として保存する．プレビューとは，カメラの映像を取り込む前に，ディスプレイに表示だけを行うことである．次のコマンドのように-t 1000 をつけると，プレビュー時間を 1 秒に設定できる．

```
$ raspistill –o image.jpg –t 1000
```

これらの命令で最大画素数のカラー画像が保存される．画像サイズを指定する場合には，次の例のように-w と-h のオプションを使う．

```
$ raspistill –o image.jpg –t 1000 –w 640 –h 480
```

ファイルに保存した画像を表示する簡単な方法は，Menu のアクセサリにあるイメージビューワを使うことである．ファイルマネージャから画像ファイルをダブルクリックしてもイメージビューワが画像を表示する．

raspistill には多くのオプションがある．Shell Script でそれらをプログラムするだけで，簡単なカメラシステムを構築することができる．オプションの一覧を表示するには，引数なしで raspistill を実行させる．

動画を撮影するには raspivid コマンドを使う．例えば，

```
$ raspivid –o video.h264 –t 10000
```

とすると，video.h264 というファイル名で，H.264 形式の動画ファイルが保存される．動画の撮影時間は-t オプションで指定しており，この場合，10 秒間撮影する．

raspivid コマンドは，1080p フォーマット（1920×1080 画素）で撮影することがデフォルトである. 画像サイズを設定するには, -w と-h のオプションを使い, 次のように記述する．

```
$ raspivid –o video.h264 –t 10000 –w 640 –h 480
```

保存した動画ファイルは，

```
$ omxplayer video.h264
```

と入力すると表示される．raspivid にも多くのオプションがある．オプションの一覧を表示するには，引数なしで raspivid コマンドを実行する．

4.3 C 言語と OpenCV による画像の入力・表示・出力

raspistill や raspivid で画像・映像の入出力は可能であるが，画像・映像の中身を処理することはできない．この実験では，C/C++言語によるプログラムで画像・映像の内容に処理を加える．カメラモジュールの画像を C/C++言語のプログラムに入力するためのライブラリとして raspicam_cv ライブラリを用いる. そして, 入力した画像をディスプレイに表示したり，処理・保存するために OpenCV を使う．これらのライブラリは，事前にインストールしてある．

図 4.2 に，カメラモジュールから画像を入力し，ディスプレイに表示し，キーボードから'q'キーを押したときに, 画像をファイルに保存して終了するプログラム raspicam.cpp の一部を示す．このプログラムは C++のコードである．C++は C 言語を含むので，自分で入力する部分は C 言語でプログラムすれば良い．見慣れない書き方もあると思うが，類推しながらプログラムを追ってほしい. 12～21 行目は raspicam_cv の機能を使うことで，カメラモジュールを変数 capture に関連付けている．24 行目は OpenCV で画像を表示するウィンドウを定義している．27 行目で画像を保存するための，cv::Mat クラスの変数 src_img を宣言する．36 行目でカメラモジュールから src_img に画像を読み込む．40 行目で，変数 src_img に読み込んだ画像をウィンドウに表示する．43～45 行目はキー入力を 1 ミリ秒待ち，その間に'q'のキ

ーが入力されれば，src_img に読み込んだ画像を src_img.jpg として保存し，プログラムを終了する．ここで注意を要することは，cv::imshow が画像を表示する関数であるが，実際にディスプレイに表示されるのは cvWaitKey 関数を実行したときになる，ということである．また 48～53 行目では，このプログラムのフレームレート（１秒あたりの処理枚数）を計算して表示している．

このプログラムの中には，これまでに見たことがない関数やデータ型，構文があると思う．それらの意味は，関数名や変数名から推察してほしい．わかりにくいものや，プログラムを作成する上で知りたいことがあれば質問すること．詳細な情報は，opencv.jp のサイトや opencv.org のサイトから得ることができる．

raspicam.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
10 int main(int argc, char *argv[]) {
11
12  //カメラモジュールを利用できるようにするための初期設定
13  RASPIVID_CONFIG *config = (RASPIVID_CONFIG*)malloc(sizeof(RASPIVID_CONFIG));
14  config->width = 640;//水平画素数
15  config->height = 480;//垂直画素数
16  config->bitrate = 0;
17  config->framerate = 0;
18  config->monochrome = 0;//0:カラー画像、1:濃淡画像
19  RaspiCamCvCapture *capture =
20    (RaspiCamCvCapture *) raspiCamCvCreateCameraCapture2(0,config);
21  free(config);
22
23  //画像を表示するためのウインドウを定義する
24  cv::namedWindow("capture", CV_WINDOW_AUTOSIZE);
25
26  //画像を保存するための変数
27  cv::Mat src_img;
28
29  //フレームレートを測るための準備。
30  struct timeval start,end;
31  gettimeofday(&start,NULL);
32  float fnum = 0.0f;//フレーム数
33
34  while (true) {
35    //カメラモジュールから画像を取り込む
36    src_img = raspiCamCvQueryFrame(capture);
37
38    //その画像をウインドウに表示する。
39    //但し，ディスプレイへの表示はcvWaitKey()関数を実行したときに行われる
40    cv::imshow("capture", src_img);
41
42    //'q'キーが押されれば、ファイルを保存して終了する
43    if (cvWaitKey(1) == 'q') {
44      imwrite("src_img.jpg",src_img);
45      break;
46    }
47
48    //フレームレート（1秒あたりの処理枚数）を計算して、表示する。
49    gettimeofday(&end,NULL);
50    float interval = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec
51    fnum += 1.0f;
52    float fps = fnum/interval;
53    printf("%g fps\n",fps);
54  }
```

図 4.2 画像を入力し，表示し，ファイルに保存するプログラム raspicam.cpp

OpenCV のような大規模なライブラリを使って，プログラムをコンパイルし実行ファイルを作るには，長いオプションのついた g++コマンドを入力する必要がある（g++は C++のコンパイラ）．そのような長いコマンドラインは入力を間違いやすいので，コマンドラインの文字列からなるテキストファイルを作成し，それに実行権限を与えることで，ファイル名を入

力するだけで複雑なコマンドラインを実行させることがある．さらに進んだ方法としてとして make コマンドがある．これは，Makefile という名前のテキストファイルに，コンパイル手順を記述しておき，make コマンドを入力することで，Makefile ファイルを実行させるものである．図 4.2 のプログラム raspicam.cpp をコンパイルするための Makefile を図 4.3 に示す．これを Makefile という名称のテキストファイルとして準備しておき，

```
$ make raspicam
```

と入力すると，図 4.2 のプログラム raspicam.cpp がコンパイルされる．ソースコードか Makefile にエラーがあれば，エラーメッセージが表示され，実行コードは生成されない．実行コード raspicam が生成されれば

```
$ ./raspicam
```

と実行して，プログラムの動作を確認すること．

Makefile

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
1  CFLAGS_OPENCV = -I/usr/include/opencv2
2  LDFLAGS_OPENCV = \
3      -lopencv_highgui -lopencv_core -lopencv_imgproc \
4      -lopencv_objdetect -lopencv_video -lpthread
5
6  USERLAND_ROOT = $(HOME)/git/raspberrypi/userland
7  CFLAGS_PI = \
8      -I$(USERLAND_ROOT)/host_applications/linux/libs/bcm_host/include \
9      -I$(USERKAND_ROOT)/host_applications/linux/apps/raspicam \
10     -I$(USERKAND_ROOT) \
11     -I$(USERKAND_ROOT)/interface/vcos/pthreads \
12     -I$(USERKAND_ROOT)/interface/vmcs_host/linux \
13     -I$(USERKAND_ROOT)/interface/mmal
14 LDFLAGS_PI = \
15     -L$(USERLAND_ROOT)/build/lib \
16     -lmmal_core -lmmal -lmmal_util -lvcos -lbcm_host
17
18 CFLAGS = $(CFLAGS_OPENCV) $(CFLAGS_PI)
19 LDFLAGS = $(LDFLAGS_OPENCV) $(LDFLAGS_PI) \
20     -lraspicamcv -lX11 -lXext -lrt -lstdc++
21
22 raspicam: raspicam.cpp
23     g++ $? $(CFLAGS) $(LDFLAGS) -O2 -o $@
```

図 4.3 raspicam.cpp をコンパイルする Makefile の例

図 4.3 の Makefile を簡単に説明する．赤い点線で囲んだ部分が OpenCV に関する設定，青い点線で囲んだ部分が raspicam_cv に関する設定，黒い点線で囲んだ部分がコンパイルの手順である．“-I”で始まる文字列はインクルードファイルのパス，“-L”で始まる文字列はライブラリファイルのパス，“-l”で始まる文字列はライブラリの名称である．22 行目の raspicam: raspicam.cpp という記述は，raspicam が raspicam.cpp から生成されることを意味している．23 行目はコンパイルのコマンドラインで，“$?”などは文字に変換される．

この実験で新しいソースプログラムを作り，それをコンパイルする場合には，Makefile 末尾の 2 行をコピペし，ファイル名を書き換えて使えば良い．Makefile と make コマンドは便利なツールであるから，使い方の雰囲気は理解すること．

4.4 OpenCV を使った画像処理

次に OpenCV を使った画像処理プログラムを説明する．画像処理の一つに線形空間フィルタがある．線形空間フィルタで実現できる処理の一つに画像の平滑化がある．これを OpenCV でプログラムすると図 4.4 のようになる．図 4.4 のプログラム image_blur.cpp は図 4.2 のプログラム raspicam.cpp に 25 行目，28 行目，38・39 行目，44 行目，49 行目を書き加えたものである．このプログラムを

```
$ make image_blur
```

とコンパイルし，続いて，image_blur を実行させて動作を確認せよ．

このプログラムを見ると，39 行目の関数 cv::blur で画像を処理している．変数 src_img が表す入力画像を平滑化し，変数 blur_img が表す画像に出力する．第三引数の cv::Size(5,5) は，5×5 画素を平均化することで平滑化することを表している．

空間フィルタと呼ばれる画像処理について調査し，処理の方法や機能について報告せよ．

image_blur.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
10 int main(int argc, char *argv[]) {
11
12   //カメラモジュールを利用できるようにするための初期設定
13   RASPIVID_CONFIG *config = (RASPIVID_CONFIG*)malloc(sizeof(RASPIVID_CONFIG));
14   config->width = 640;//水平素数
15   config->height = 480;//垂直画素数
16   config->bitrate = 0;
17   config->framerate = 0;
18   config->monochrome = 0;//0:カラー画像、1:濃淡画像
19   RaspiCamCvCapture *capture =
20     (RaspiCamCvCapture *) raspiCamCvCreateCameraCapture2(0,config);
21   free(config);
22
23  //画像を表示するためのウインドウを定義する
24  cv::namedWindow("capture", CV_WINDOW_AUTOSIZE);
25  cv::namedWindow("blur", CV_WINDOW_AUTOSIZE);
26
27  //画像を保存するための変数
28  cv::Mat src_img,blur_img;
29
30  //フレームレートを測るための準備。
31  struct timeval start,end;
32  gettimeofday(&start,NULL);
33  float fnum = 0.0f;//フレーム数
34
35  while (true) {
36    //カメラモジュールから画像を取り込む
37     src_img = raspiCamCvQueryFrame(capture);
38    //画像を、5x5の近傍画素を平均化することで平滑化する
39    cv::blur(src_img,blur_img,cv::Size(5,5));
40
41    //その画像をウインドウに表示する。
42    //但し，ディスプレイへの表示はcvWaitKey()関数を実行したときに行われる
43     cv::imshow("capture", src_img);
44     cv::imshow("blur", blur_img);
45
46     //'q'キーが押されれば、ファイルを保存して終了する
47     if (cvWaitKey(1) == 'q') {
48       imwrite("src_img.jpg",src_img);
49       imwrite("blur_img.jpg",blur_img);
50       break;
51     }
52
53    //フレームレート（1秒あたりの処理枚数）を計算して、表示する。
54     gettimeofday(&end,NULL);
55     float interval = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec
56     fnum += 1.0f;
57     float fps = fnum/interval;
58     printf("%g fps\n",fps);
59  }
```

図 4.4 OpenCV を使って画像の平滑化を行うプログラム image_blur.cpp

OpenCV を使った画像処理プログラムの別の例を図 4.5 に示す．これは，入力画像からエッジを検出するプログラムである．39 行目の関数 cv::cvtColor で入力のカラー画像 src_img を濃淡画像 gray_img に変換し，41 行目の関数 cv::Canny でその濃淡画像からエッジ画像 edge_img を計算している．このように OpenCV の関数を書き並べるだけで，複雑な処理を実現することができる．

エッジ抽出と呼ばれる画像処理について調査し，処理の方法や用途について報告せよ．

image_edge.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
23  //画像を表示するためのウインドウを定義する
24   cv::namedWindow("capture", CV_WINDOW_AUTOSIZE);
25   cv::namedWindow("edge", CV_WINDOW_AUTOSIZE);
26
27  //画像を保存するための変数
28   cv::Mat src_img,gray_img,edge_img;
29
30  //フレームレートを測るための準備。
31   struct timeval  start,end;
32   gettimeofday(&start,NULL);
33   float fnum = 0.0f;//フレーム数
34
35   while (true) {
36    //カメラモジュールからカラー画像を取り込む
37     src_img = raspiCamCvQueryFrame(capture);
38    //カラー画像を濃淡画像に変換する
39     cv::cvtColor(src_img,gray_img,CV_RGB2GRAY);
40     //Cannyオペレータでエッジ画像を求める
41     cv::Canny(gray_img, edge_img, 50, 100);
42
43    //画像をウインドウに表示する。
44    //但し，ディスプレイへの表示はcvWaitKey()関数を実行したときに行われる
45     cv::imshow("capture", src_img);
46     cv::imshow("edge", edge_img);
```

図 4.5 Canny オペレータでエッジを抽出するプログラム image_edge.cpp

image_blur.cpp，image_edge.cpp 以外に，今回の実験のために，binarize.cpp，histEqualize.cpp，faceDetect.cpp，opticalFlow.cpp を作っておいた．これらは，画像の二値化，ヒストグラム平坦化，顔検出，オプティカルフローの計算を行うプログラムである．

それぞれのプログラムの具体的な動作・機能を観察して，報告せよ．

4.5 OpenCV における画像データへのアクセス

図 4.4 で使った cv::blur や図 4.5 の cv::cvtColor，cv::Canny は画像を処理する関数の一つである．OpenCV には，このような画像処理関数がたくさん準備されている．これらを組み合わせることで多くのことが実現できる．しかし場合によっては，自分で画像処理の内容をプログラムすることもある．そのような場合，画像の中身にアクセスして処理手順をプログラムする．

画像は画素が縦横に規則正しく並んだものである．その画素の値にアクセスするプログラムの例を図 4.6 に示す．図 4.6 のプログラム pixel_access.cpp は，図 4.2 のプログラム

raspicam.cpp に，全画素値を読み込み，ある処理を行い，同じ画素に描き戻す処理（図 4.6 で図示した部分）を加えている．このプログラムをコンパイルして，動作を確認せよ．プログラムを実行させる場合，

```
$ ./pixel_access 1
```

と実行させると，濃淡画像の処理になり，

```
$ ./pixel_access
```

と実行させるとカラー画像の処理になる．

このプログラムでカラー画像を読み込んだ場合，図 4.6 の青枠で囲んだ部分が実行される．一つの画素の値を r,g,b という 3 つの変数に読み込み，3 つの変数の値に~という演算を施し，それを書き戻している．r,g,b はカラー画像の画素値を表す赤・緑・青の値を unsigned char 型（8 ビットの符号なし整数，つまり，0～255 の整数）で表す変数である．画素値にアクセスするために，src_img.at<cv::Vec3b>(y,x)[0]のように書いている．src_img は入力画像を表す cv:Mat クラスの変数である．at は cv::Mat クラスに定義されている画素値をアクセスするためのメソッド（クラスに定義される関数はメソッドと呼ばれる）である．<cv::Vec3b> はアクセスするデータが cv::Vec3b という型（3 バイトのデータ列の意味）であることを示す．(y,x)[0]はアクセスする画素の場所が(y,x)で 0 番目のデータにアクセスする，という意味である．濃淡画像を読み込んだ場合の画素値へのアクセスは赤枠で囲んだように行う．

48～69 行目のプログラムと観察した処理画像の関係について説明せよ．

*pixel_access2.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
41    while (true) {
42       //カメラモジュールから画像を取り込む
43       src_img = raspiCamCvQueryFrame(capture);
44       reverse_img = cv::Mat(src_img.size(),src_img.type());//反転画像を保存する領域を確保する
45
46       //画素値へのアクセス
47       int y,x;//画素の位置（画像座標）
48       if (monochrome == 1) {//濃淡画像の場合の処理
49          unsigned char gray;//濃淡画像の画素値
50          for (y = 0; y < src_img.rows; y++) {
51             for (x = 0; x < src_img.cols; x++) {
52                gray = src_img.at<unsigned char>(y,x);
53                reverse_img.at<unsigned char>(y,x) = ~gray;
54             }
55          }
56       } else {//カラー画像の場合の処理
57          unsigned char r,g,b;//カラー画像の画素値
58          for (y = 0; y < src_img.rows; y++) {
59             for (x = 0; x < src_img.cols; x++) {
60                b = src_img.at<cv::Vec3b>(y,x)[0];
61                g = src_img.at<cv::Vec3b>(y,x)[1];
62                r = src_img.at<cv::Vec3b>(y,x)[2];
63
64                reverse_img.at<cv::Vec3b>(y,x)[0] = ~b;
65                reverse_img.at<cv::Vec3b>(y,x)[1] = ~g;
66                reverse_img.at<cv::Vec3b>(y,x)[2] = ~r;
67             }
68          }
69       }
```

図 4.6 画素値のビットパターンを反転させるプログラム pixel_access.cpp

4.6 実験 2 の作成課題

4.6.1 作成課題 1、画像の平滑化

pixel_access.cpp を参考にして，図 4.4 の 39 行目で画像を平滑化している部分（関数 blur の部分）を，自作のプログラムコードによって実現せよ．行うことは概ね，39 行目をコメントアウトし，そこに図 4.6 の 58～68 行目のような，画素値を処理するコードを書くことである．次のことに注意すること．

① 全ての画素について，その画素を中心として上下左右の 25 画素の値を平均化したものを新しい画素値とする．

② ただし画像の境界から 2 画素の範囲にある画素は，①のように計算できないので，今回は計算しないでよい．

③ 作成するプログラムでは，図 4.4 の 29 行目の位置に blur_img = cv::Mat(480, 640, CV_8UC3);の行を追加すること．これは縦 480×横 640 画素のカラー画像の領域を変数 blur_img に設定するという意味になる．

④ 入力画像の画素値は符号なし 8 ビット（0～255 の値）で保存されているが，それを足し合わせると 255 を超えることがある．

4.6.2 作成課題 2、エッジ抽出

image_edge.cpp を参考にして，図 4.5 の 41 行目でエッジ抽出を行っている部分（関数 Canny の部分）を，自作のコードによって実現せよ．ただし，Canny 関数は高度な処理を行っているので，それを忠実に実装することは難しい．この実験では，次のような簡単な方法を実装すれば良い．

① 全ての画素について，その画素位置における濃度の勾配（画素値の変化率）を求め，それがある閾値よりも大きいものをエッジとして抽出する．エッジと判断した画素の値は 255 にし，そうでない画素の値は 0 とする．

② 注目画素の濃度勾配は，右隣の画素との差の二乗と直下の画素との差の二乗を加算し，平方根（ルート）をとったものとする．

③ 隣接する画素の差を計算するので，画像の右端と下端の画素はエッジを抽出することができない．

④ 作成するプログラムでは，図 4.5 の 29 行目の位置に edge_img = cv::Mat(480, 640, CV_8UC1);の行を追加すること．これは縦 480×横 640 画素の濃淡画像の領域を変数 edge_img に設定するという意味になる．

自作のエッジ抽出プログラムが動作すれば，その処理結果を image_edge.cpp の処理結果と比較して，どのような違いがあるかを考察せよ．

以上で実験 2 を終了する．

4.7 実験 2 のレポートについてのアドバイス

タイトル：「OpenCV を使った画像処理」

概要：最初に 300 文字程度で概要（この実験の成果を短くまとめたもの）を書くこと．

本文：次のような章立てで記述すること．具体的な章・節のタイトルは自分で考えること．また，自分の考えがあれば，章立てを変更しても良い．この資料で，青字で示した部分に対しては，レポート内に解答を記述すること．

1 章　この実験の目的を書く．
2 章　実験の前提になっていることについて書く．カメラモジュールの性能や画像をプログラムに取り込むための方法など．
3 章　プログラムを作成する前に行った検討内容について書く．OpenCV による画像処理の方法や，画素データにアクセスする方法など．
4 章　作成したプログラムについて書く．どのように問題を解決したか．解決できなかった問題はなにか，などを書く．
5 章　まとめる

本文中で指示したことに従うこと．「CSE 実験レポートの書き方」を参考にして，わかりやすい日本語で書くこと．適切に図表を挿入すること．

5 章 実験 3：加速度・角速度センサを使った位置と姿勢の計測

この実験では加速度･角速度センサを Raspberry Pi に接続し,その信号を処理することで,物体の位置と姿勢を計測する.

5.1 加速度・角速度センサ

実験 3 では，加速度・角速度センサ（GY-521，図 5.1 のもの）を使って，物体の位置と姿勢を計測する．図 5.1 のプリント基板中央にある 4mm 四方の部品が，3 軸周りの角速度と 3 軸方向の加速度を計測するセンサ（MPU6050）である．なお，角速度センサはジャイロとよぶことも多い.航空機などの姿勢を制御するために用いるジャイロスコープを起源としている.もともとは比較的，大きな装置であったが，現在では，MEMS（Micro Electro Mechanical Systems）技術を用いることで，加速度センサと角速度センサを一体化し，非常に小型に作ることができる.

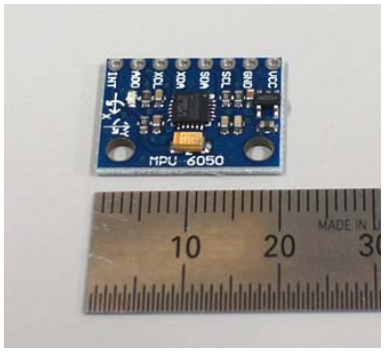


図 5.1 ３軸ジャイロ・加速度センサモジュール（GY-521）

5.2 位置・姿勢と加速度・角速度の関係

３次元空間に存在する物体の位置と姿勢は３次元座標で定義する．３次元座標の位置と姿勢は３要素の並進ベクトル（位置）と，３つの回転角度（姿勢）で指定する（図 5.2 左）．図 5.1 のセンサで３次元空間の位置と姿勢を計測することが可能であるが，この実験では，簡単化のために，一つの軸方向の位置と姿勢だけを扱う（図 5.2 右）.

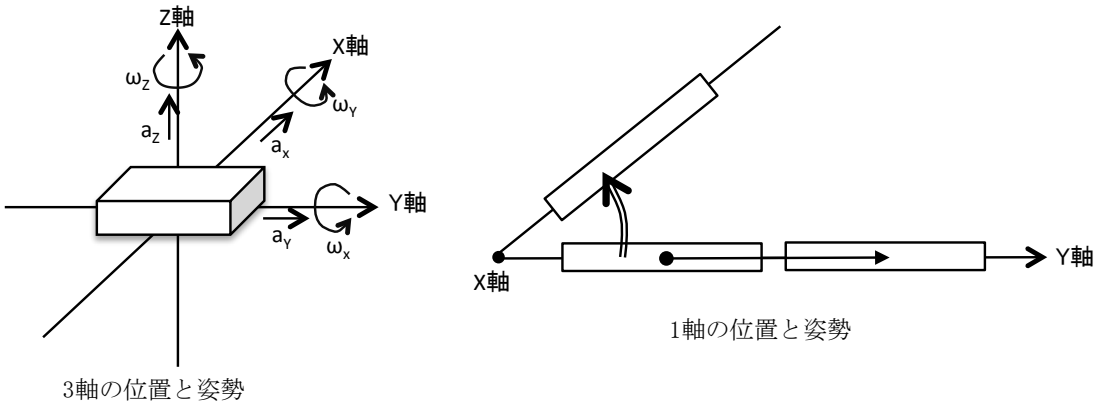


図 5.2 ３次元空間における加速度と角速度

図 5.2 右は，図 5.2 左の様子を X 軸の負の方向から見たものである．この実験では，この図が示すように，Y 軸方向の位置と X 軸まわりの姿勢（回転角）だけを扱う．位置は軸方向の移動量で定義し，姿勢は軸周りの回転角で定義する．

時刻 0 に物体が静止しており，その後，時刻 t における Y 軸方向の加速度が $a(t)$であったとすると，時刻 t における物体の速度 $v(t)$は，$v(t)=\int_0^t a(t)dt$ となる（加速度を時間について積分すると速度になる）．そして，時刻 t における物体の位置 $p(t)$は，$p(t)=\int_0^t v(t)dt$ になる（速度を時間について積分すると移動量（位置）になる）．同様に，時刻 t における X 軸周りの角速度が $\omega(t)$であるとき，時刻 t における物体の姿勢を表す角度 $\theta(t)$は $\theta(t)=\int_0^t \omega(t)dt$ となる（角速度を時間について積分すると角度になる）．このように，加速度と角速度を時間に対する連続量 $a(t)$，$\omega(t)$で表現し，それらから速度や位置，角度を求めるには，時間について積分する必要がある．

一方，加速度・角速度センサをコンピュータに接続し，離散的な時間間隔でコンピュータが読み取るセンサ値から速度や位置，角度を求める計算は，上記の積分を数値的に計算するものである（数値積分という）．詳細は 5.4 節で説明するが，要するに，加速度センサの出力値と時間間隔の乗算で，その時間間隔での速度変化を求め，時間間隔ごとの速度変化を積算すること速度を求める．角速度について言えば，角速度センサの出力値と時間間隔の乗算で，その時間間隔での角度変化を求め，時間間隔ごとの角度変化を積算することで角度を求める．

ところで，図 5.1 のセンサの X 軸・Y 軸はプリント基板にシルク印刷されている（シルク印刷とは，プリント基板上に描かれている印刷のこと）．Y 軸はセンサ基板の縦方向，X 軸は横方向である．Z 軸はプリント基板に垂直な軸になる．

5.3 センサ信号を解析する

センサと RasPi2 をブレッドボードとジャンパ線を用いて図 5.3 のように接続する．RasPi２にセンサを接続する時には，必ず，RasPi2 の電源を切っておくこと．センサと RasPi2 は，I^2C（Inter-Integrated Circuit）と呼ばれるシリアルバスで通信を行う．I^2C はさまざまなデバイスとデジタル装置の間で通信を行うための規格である．RasPi2 で I^2C を使えるようにするには設定が必要であるが，この実験の RasPi2 は設定ずみである．

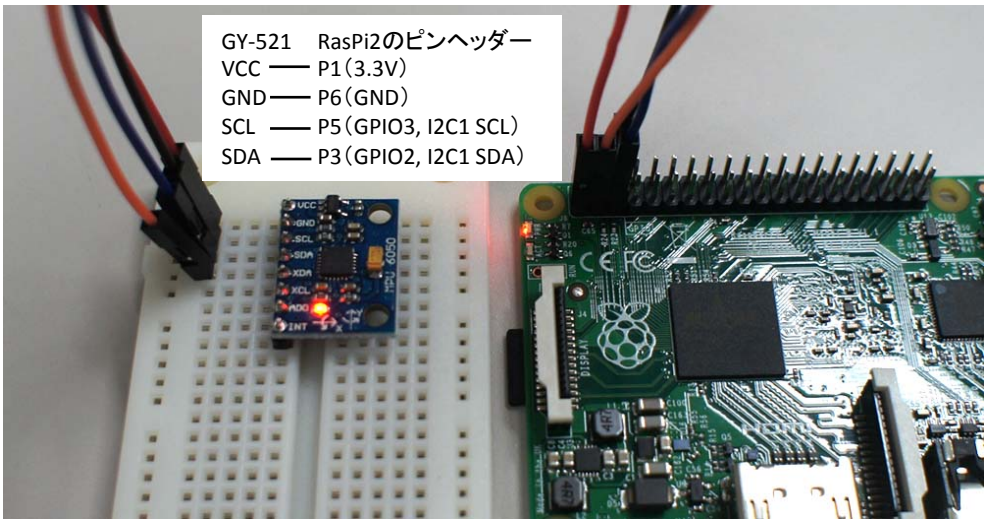


図 5.3 加速度・角速度センサと RasPi2 の接続

図 5.4 のプログラム mpu.c を使って Y 軸方向の加速度と X 軸周りの角速度を入力する．図 5.4 の 15～18 行目はセンサチップである MPU6050 の機能にアクセスするためのアドレスである．29 行目で Y 軸方向の加速度，32 行目で X 軸周りの角速度を入力し，34 行目でそれらを printf する．加速度 ay は重力加速度を単位とする数値で，角速度 gx はラジアン/s を単位とする数値で表示される．図 5.5 のプログラム mpu.c から

```
$ gcc -o mpu mpu.c -lwiringPi
```

によって実行ファイル mpu を作る．

mpu を実行すると加速度 ay の値と角速度 gx の値が表示される．センサが乗ったブレッドボードを傾けたり動かすと，表示される数値が変化する．例えば，Y 軸方向を垂直に立てると ay が 1.0 あるいは-1.0 前後の数値になる．これは Y 軸方向に重力加速度が加わるからである．また，X 軸方向を軸にして回転させると gx の数値の絶対値が大きくなる．これは，X 軸周りに大きな角速度が発生するためである．

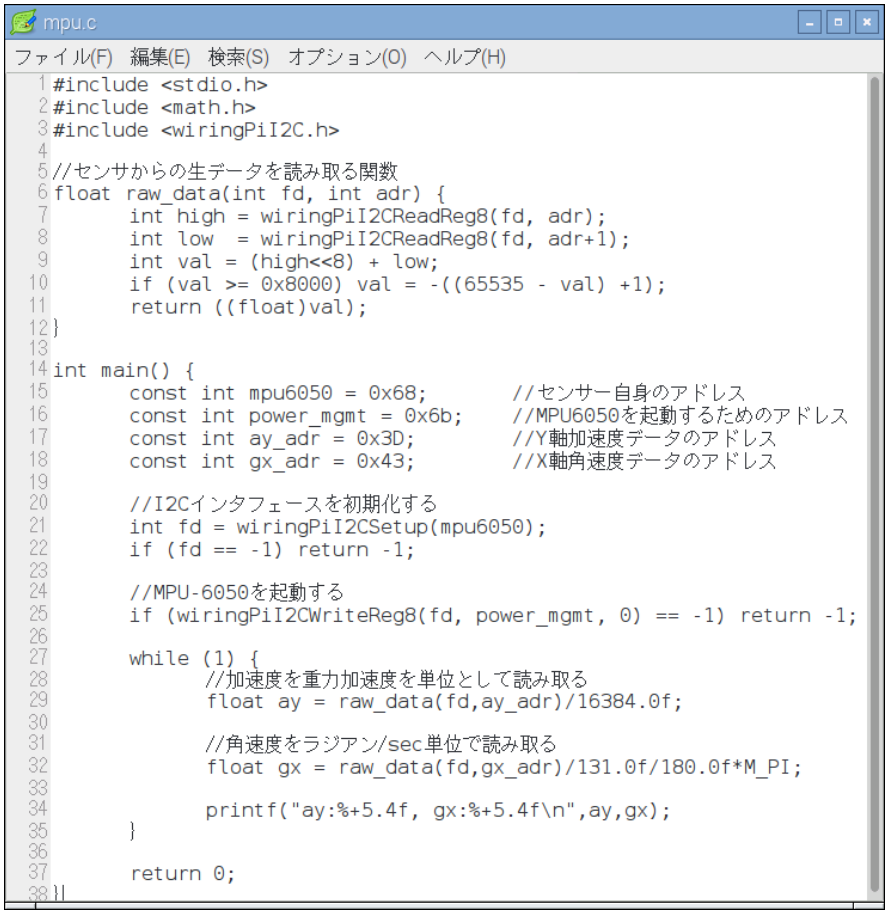


図 5.4 加速度・ジャイロセンサから信号を読み取るプログラム mpu.c

作成したプログラム mpu.c とセンサを装着したブレッドボードを利用して，センサの位置・姿勢や動きとセンサから得る数値の関係がどのようになっているかを詳しく知ろう．そのために，次のことについて実験し，報告せよ．

① ブレッドボードを机の上に静止させた状態では，ay，gx ともに 0 に近い値である．しかし，それらの数値は揺らいでいるし，また，完全に 0 ではない．まず，センサの出力値のそれぞれについて平均値を求めよ．平均値を求めるためのサンプル数は，十分に多く取る必要がある．今回は 10,000 個にする．これを行うために，mpu.c のプログラムを参考にして，それにデータの平均値を求めるコードを追加すること．（注：センサーデータを 10,000 個読み込むと 10 秒程度の時間を要する．この程度の時間であっても，デバッグのために何度も繰り返すと，時間を無駄にしているように感じる．そこで，繰り返し回数を記号定数として#define で定義し，それを利用したプログラムを作成するとよい．デバッグ中は 100 回程度の繰り返し回数で実行させ，プログラムを完成させる段階で 10,000 回に設定する）

② 次に，測定値の揺らぎの程度を計算しよう．一定の値であるべきセンサの出力値がラン

ダムに微小変化する現象を，測定値にノイズがのっているという．この実験では，ノイズの量を平均値との差によって評価するために，分散と標準偏差を計算する．2 つの測定値について，分散と標準偏差を報告せよ．分散・標準偏差を求めるためのサンプル数は，10,000 個にすること．①と同様に，mpu.c のプログラムにコードを追加すること．

平均値，分散，標準偏差を計算する式は次のようになる．ここで，data(i)が測定値，i=1～N，N は測定値の個数である．

平均： $\mu = \frac{1}{N}\sum_{i=1}^{N} data(i)$

分散： $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(data(i)-\mu\right)^2$

標準偏差： $\sigma = \sqrt{\sigma^2}$

次の実験③に行く前に，計算した平均・分散・標準偏差の値が，概ね正しいことは確認せよ．どのように判断したかを報告せよ．

一般に，センサを同じ状態に保って測定を繰り返すと，個々の測定値はばらつくが，測定値の平均は一つの値に収束する．ばらつき具合を評価するために，測定値のヒストグラム（分布状態）をとると，図 5.5 の形になることが多い．この図は横軸を測定値，縦軸を測定値の頻度（回数，分布密度）としたものであり，曲線の形状は平均値 μ ・標準偏差 σ の正規分布（ガウス分布ともいう）である．測定値の分布が正規分布にしたがうならば，全データの 68.26%は $\mu \pm \sigma$ に収まり，全データの 99.74%は $\mu \pm 3\sigma$ に収まることを示している．

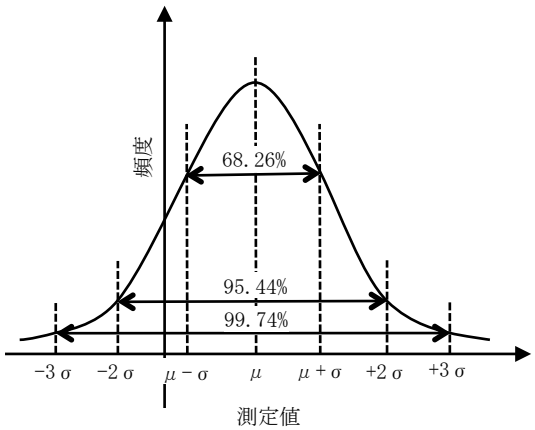
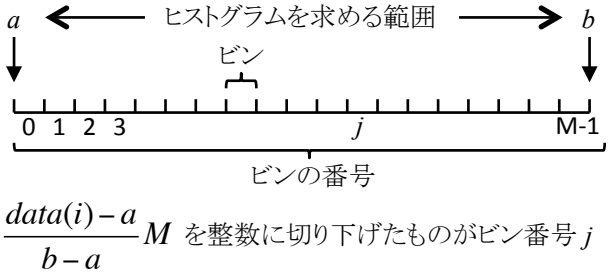


図 5.5 ランダムなノイズをもつ測定値の分布（正規分布（ガウス分布））

③ 10,000 個の測定データのヒストグラムを求めよう．測定データが正規分布しているなら，その 99.74%，すなわち 9,974 個は $\mu \pm 3\sigma$ の範囲に収まっているはずである．ヒストグラムを計算するにあたって，念のために $\mu \pm 4\sigma$ の範囲で考える．この範囲を幅一定の 100 個の区間に分割し，各区間に存在する測定値の個数をカウントするプログラムを作成せ

よ．100 個の数値はテキストファイルにして持ち帰り，Excel などのソフトウエアを使ってグラフ化すること．そのグラフは図 5.5 のようになるので（折れ線グラフの場合は図 5.5 のようになる．棒グラフでもよい.），それをレポートに貼り付けること．

ヒストグラムの計算法を，図 5.6 を使って説明する．ヒストグラムを計算する範囲を a〜b とする．この範囲を M 個の区間に分解し，それぞれに区間に属するデータの個数を求めることを考える.それぞれの区間をヒストグラムのビンとよぶ.全体で M 個のビンがある.data(i) が属するビンの番号は図 5.6 に書いたように求めることができる．



$\frac{data(i)-a}{b-a}M$ を整数に切り下げたものがビン番号 j

図 5.6 *data*(*i*)が属するビンの求め方

5.4 センサ信号を処理する

センサ値を入力し始めた時刻を t_0 とし，この時点で加速度が 0m/sec^2，角速度が 0 ラジアン/sec であったとする．そして，時刻 t_i における加速度を $a(t_i)$，角速度を $\omega(t_i)$とする．加速度と速度・位置の関係は 5.2 節で説明したとおりである．これをプログラムで計算する場合，表 5.1 のように，加速度と時間変化の積から速度の変化量を求め，それを足し合わせることで速度を推定する．そして，速度と時間変化の積から位置の変化量を求め，それを足し合わせることで位置を推定する．角速度と角度の関係も同様で，角速度と時間変化の積を足し合わせることで角度を推定する．

表 5.1 時間の加速度・速度・位置，角速度・角度の関係

時刻:t	t_0	t_1	t_2	…	t_n
時間変化	0	t_1-t_0	t_2-t_1	…	t_n-t_{n-1}
加速度:a	0	$a(t_1)$	$a(t_2)$	…	$a(t_n)$
速度:v	$v(t_0)$=0	$v(t_1)$= $v(t_0)+a(t_1)(t_1-t_0)$	$v(t_2)$= $v(t_1)+a(t_2)(t_2-t_1)$	…	$v(t_n)$= $v(t_{n-1})+a(t_n)(t_n-t_{n-1})$
位置:p	$p(t_0)$=0	$p(t_1)$= $p(t_0)+v(t_1)(t_1-t_0)$	$p(t_2)$= $p(t_1)+v(t_2)(t_2-t_1)$	…	$p(t_n)$= $p(t_{n-1})+v(t_n)(t_n-t_{n-1})$
角速度:ω	0	$\omega(t_1)$	$\omega(t_2)$	…	$\omega(t_n)$
角度:θ	$\theta(t_0)$=0	$\theta(t_1)$= $\theta(t_0)+\omega(t_1)(t_1-t_0)$	$\theta(t_2)$= $\theta(t_1)+\omega(t_2)(t_2-t_1)$	…	$\theta(t_n)$= $\theta(t_{n-1})+\omega(t_n)(t_n-t_{n-1})$

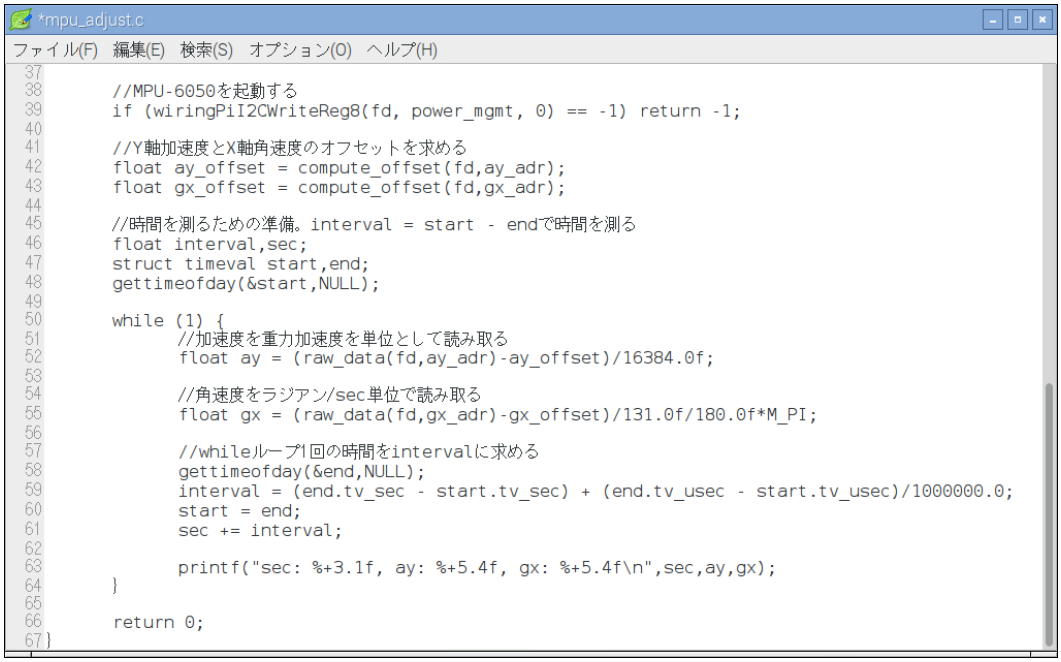
センサ信号の扱いでは，オフセットとドリフトがしばしば問題になる．オフセットは，ゼロであるべき信号値が，定常的にずれている状態である．ドリフトはセンサの特性が時間とともに変化する現象である．5.3 節で ay と gx の平均値を求めた．それらの値は，理想的にはゼロであるが，そうではなく，無視できない数値になっていることが多い．これらはオフセットである．センサの信号処理では，オフセットやドリフトをキャンセルすることが重要である．

図 5.7 に ay と gx のオフセットをキャンセルするプログラム mpu_adjust.c を示す．このプログラムは 42・43 行目でそれぞれのオフセット求めている．関数 compute_offset は，このプログラムの少し上に記述してあるのだが，1000 回の読み込み値を平均したものを返す．そして 52 行目と 55 行目で，センサの読み込み値からこれらのオフセットを引くことで，オフセットをキャンセルしている．

なお，図 5.7 のプログラム mpu_adjust.c は 45 行目～48 行目，及び，57 行目～63 行目のコードも追加されている．これらは，while ループの 1 回の繰り返しに要する時間を計るためのものである．このプログラムを

```
$ gcc -o mpu_adjust mpu_adjust.c -lwiringPi
```

とコンパイルして実行すると，センサが静止状態にあるときは，ay，gx の値が平均 0 になっていることと，実時間がとれていることがわかる．




```
37
38        //MPU-6050を起動する
39        if (wiringPiI2CWriteReg8(fd, power_mgmt, 0) == -1) return -1;
40
41        //Y軸加速度とX軸角速度のオフセットを求める
42        float ay_offset = compute_offset(fd,ay_adr);
43        float gx_offset = compute_offset(fd,gx_adr);
44
45        //時間を測るための準備。interval = start - endで時間を測る
46        float interval,sec;
47        struct timeval start,end;
48        gettimeofday(&start,NULL);
49
50        while (1) {
51              //加速度を重力加速度を単位として読み取る
52              float ay = (raw_data(fd,ay_adr)-ay_offset)/16384.0f;
53
54              //角速度をラジアン/sec単位で読み取る
55              float gx = (raw_data(fd,gx_adr)-gx_offset)/131.0f/180.0f*M_PI;
56
57              //whileループ1回の時間をintervalに求める
58              gettimeofday(&end,NULL);
59              interval = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec)/1000000.0;
60              start = end;
61              sec += interval;
62
63              printf("sec: %+3.1f, ay: %+5.4f, gx: %+5.4f\n",sec,ay,gx);
64        }
65
66        return 0;
67  }
```

図 5.7 オフセットをキャンセルしてセンサ値を読み取るプログラム mpu_adjust.c

5.5 実験 3 の作成課題

mpu_adjust.cpp を参考にして，角速度センサの信号値から角度を計算するプログラムを作成せよ．角度は度で表示せよ．なお，mpu_adjust.cpp の gx はラジアン/sec を単位としているが，ラジアンを度に変換するには M_PI で割り，180 をかける．

プログラムが正しく動作するようになれば，オフセットをキャンセルする機能がどの程度，有効に働いているかを確認して報告せよ．オフセットをキャンセルする場合としない場合の動作を，できる限り正確に記述せよ．

以上で実験 3 を終了する．

5.6 実験 3 のレポートについてのアドバイス

タイトル：「角速度・加速度センサを使った位置と姿勢の計測」

概要：最初に 300 文字程度で概要（この実験の成果を短くまとめたもの）を書くこと．

本文：次のような章立てで記述すること．具体的な章・節のタイトルは自分で考えること．また，自分の考えがあれば，章立てを変更しても良い．この資料で，青字で示した部分に対しては，レポート内に解答を記述すること．

1 章	この実験の目的を書く
2 章	実験の前提になっていることについて書く．角速度・加速度センサの一般的な性質や，角速度と角度の関係，加速度と速度・位置の関係など．
3 章	プログラムを作成する前に行った検討内容について書く．センサ信号の解析結果や，位置と姿勢を得るためのセンサ信号の処理方法など．
4 章	作成したプログラムについて書く．どのように問題を解決したか．未解決で残った問題はなにかなどを書く．
5 章	まとめる

本文中で指示したことに従うこと．「CSE 実験レポートの書き方」を参考にして，わかりやすい日本語で書くこと．適切に図表を挿入すること．

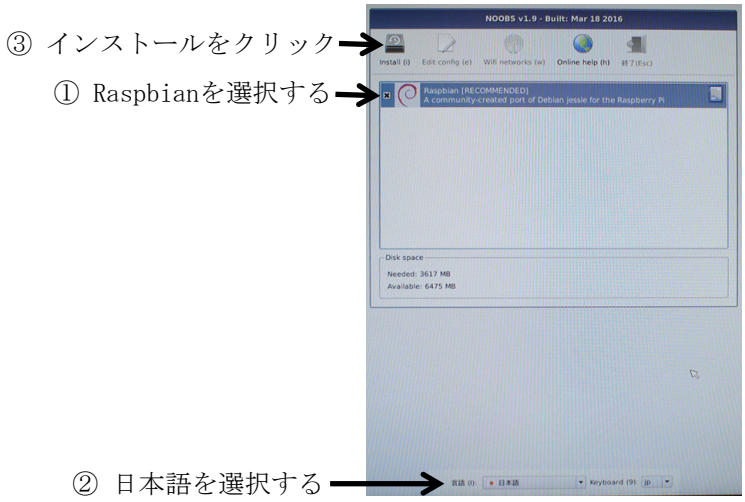
付録 1：Raspbian のインストールから日本語環境の設定まで

2017 年 3 月 13 日に，NOOBS_v2_3_0 をダウンロードして，Raspbian をインストールした手順を説明する（インストール画面と操作は，バージョンによって若干の違いがある）．

インストールに先立って SD カードをフォーマットする．フォーマッターは SD アソーシエーションから提供されている(https://www.sdcard.org/jp/downloads/formatter_4/)．フォーマットにはクイックフォーマットと上書きフォーマットがある．通常，クイックフォーマットで良いが，カードに残った情報を読み取られる可能性がある．

Raspberry Pi の DOWNLOADS サイト（http://www.raspberrypi.org/downloads/）から NOOBS の Download ZIP をダウンロードする．解凍した NOOBS_v2_2_0 フォルダの中身を SD カードにコピーする（フォルダごとコピーしないで，中身をコピーすること）．

SD カードを Raspberry Pi に装着し，ディスプレイ・キーボード・マウスを接続し，電源を投入する．しばらくすると，付録 1-図 1 に示す，インストール OS を選択する画面になる．Raspbian と日本語を選択し，インストールをクリックする．20 分程度で，「OS のインストールに成功しました」と表示されるので OK をクリックする．



付録 1-図 1 OS の選択画面

続いて Raspbian が立ち上がり，デスクトップ画面が表示される．ここでインターネットに接続し，ターミナルから次のコマンドを入力することでシステムを更新する。

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

なお，京産大の学内から有線でインターネットに接続する場合，Web ブラウザから

https://ccauth.kyoto-su.ac.jp

にアクセスし，認証を受ける必要がある．

ここから次の順番で日本語環境を整える．

① キーボードの設定

② 日本語フォントとかな漢字変換ソフトをインストール

③ ロケールとタイムゾーンの設定

ターミナルを起動し，

```
$ sudo raspi-config
```

と入力する．raspi-config はシステムの設定を行うプログラムである．Localisation Options を選択し，続いて，Change Keyboard Layout を選択する．キーボードモデルの選択で「標準 101 キー PC」，キーボードのレイアウトで「日本語」，AltGr として機能させるキーは「キーボード配置のデフォルト」，コンポーズキーは「コンポーズキーなし」，X サーバを強制終了するのに Control+Alt+Backspace を使いますか？に「はい」を選択する．（なおインストール直後であれば，「」内の文字は，英語で表示される．また，使用するキーボードによっては，異なる選択を行う必要がある．）以上で，記号キーが正しく入力できるようになる．

日本語の表示と入力を行うために，日本語フォント・入力メソッド・かな漢字変換ソフトをインストールする．

```
$ sudo apt-get install ttf-kochi-gothic xfonts-intl-japanese xfonts-intl-japanese-big xfonts-kaname
$ sudo apt-get install uim uim-anthy
```

uim（Universal Input Method）は多言語入力メソッドフレームワーク，uim-anthy はかな漢字変換ソフトウエアである．

最後にロケールとタイムゾーンを設定する．raspi-config を起動し，Localisation Options を選択し，続いて，Change Locale を選択する．ロケールの選択画面で「en_GB.UTF-8 UTF-8」「ja_JP.EUC-JP EUC-JP」「ja_JP.UTF-8 UTF-8」の三箇所でスペースキーを押す．＊マークが表示され，これらの設定が選択されたことを示す（すでに＊マークが表示されておれば，そのままでよい）．<了解>し，続くデフォルトロケールの設定画面では「ja_JP.UTF-8」を選択し，<了解>する．raspi-config の初期画面に戻るので，Localisation Options を選択し，Change Timezone を選択する．地理的領域で「アジア」を選択する．時間帯で「TOKYO」を選択する．

これらの設定はリブートすることで有効になる．以上．

これらの設定の順番を間違えると日本語が文字化けする．一旦，文字化けすると，なにを操作しているのかわからなくので，慎重に行うこと．

2017 年 3 月 13 日更新

付録 2：テキストエディタ

Raspbian に最初からインストールされているエディタは vi , nano，LeafPad である．次のコマンドで emacs をインストールする．

```
$ sudo apt-get install emacs
```

インストールを終了すると，Menu のアクセサリに GNU Emacs が追加されている．Emacs を起動し，日本語変換などのキー入力が正しく行えることを確認する．（過去に，emacs でだけ，文字化けすることがあった．）

2017 年 3 月 13 日更新

付録 3：Raspberry Pi 用 SD カードのバックアップと複製

SD カードに Raspbian をインストールし，また，raspi-config の設定や必要なソフトウエアのインストールを行ったカードの内容をバックアップする方法と，同じ内容の SD カードを複製する方法を説明する．ここでは Mac で行う場合について説明する．

複製元の SD カードをカードリーダーに接続する．デスクトップに RECOVERY と boot の 2 つのボリュームが表示される．ターミナルを起動し，

```
$ mount
```

を入力する．出力メッセージの中に

/dev/disk2s6 on /Volumes/boot …

/dev/disk2s1 on /Volumes/RECOVERY …

の表示が見える．“disk2”の部分は Mac の機種や SD カードリーダーによって異なる可能性がある．この表示は，SD カードが/dev/disk2 としてマウントされていることを示す．次に

```
$ diskutil　umountDisk　/dev/disk2
$ sudo　dd　if=/dev/rdisk2　of=ファイル名.dmg　bs=1m
```

として，SD カードの内容をファイル名.dmg にコピーする．dd はファイルのコピーと変換を行う Linux コマンドである．/dev/rdisk2 と’r’が付いているのはアンバッファモードでの転送を意味し，転送が高速になる．bs=1m は，転送時のブロックサイズが 1MB であることを意味する．8GB・class10 の SD カードの読み込みに 7 分程度を要した．

```
$ diskutil　eject　/dev/disk2
```

を入力した後，SD カードを取り出す．

コピー先のフォーマット済み SD カードをスロットに接続する．

```
$ diskutil　umountDisk　/dev/disk2
$ sudo　dd　if=ファイル名.dmg　of=/dev/rdisk2　bs=1m
```

として，ファイル名.dmg を SD カードに書き込む．8GB・class10 の SD カードで書き込みに 14 分程度を要した．書き込み終了後，

```
$ diskutil　eject　/dev/disk2
```

とした後，SD カードを取り出す．

バックアップした SD カードを他の SD カードに書き戻す時，SD カードの容量は正確に一致している（あるいは大きい）必要がある．同じ容量の SD カードであっても，製品が異なるとわずかの容量差があり，小さい SD カードへの複製は失敗する．SD カードのメモリ容量は，ディスクユーテリティを使えば，バイト単位で確認することができる．

2017 年 3 月 2 日更新

付録 4：USB メモリの利用

USB メモリを挿入すると，ファイルマネージャで開くかどうかの確認画面が現れる．OK をクリックすると，ファイルマネージャを通じて USB メモリにアクセスすることができる．また，/media/pi/の下に USB メモリの名前の付いたディレクトリができるので，これを介してアクセスすることができる．

USB メモリを取り外す場合，まず，/media/pi/に 1DEF-5904 のような名前のディレクトリがあることを確認する．具体的な名前は，USB メモリごとに異なる．そして，

```
$ umount　/media/pi/1DEF-5904
```

を実行し，USB メモリをアンマウントし，取り外す．あるいは，画面右上，タスクバーの右端にある△のアイコンクリックし，該当する USB メモリを選択する．その後，取り外す．

2017 年 3 月 2 日更新

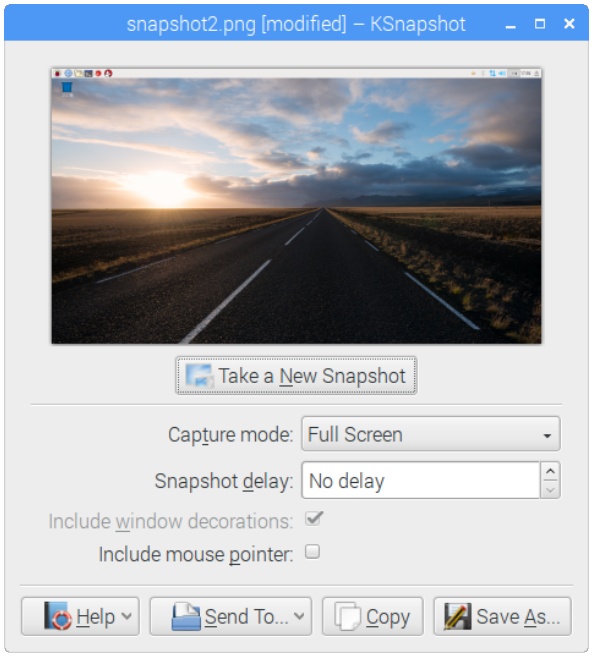
付録 5：スクリーンショットの撮り方

スクリーンショットを撮るツール KSnapshot について説明する．

```
$ sudo apt-get install ksnapshot
```

でインストールする（実験で使う Raspberry Pi にはインストール済み）．

例えば，Menu から Run を選び，ksnapshot を実行して起動する．付録 5-図 1 の操作画面が表示される．Capture mode でスクリーンショットの撮り方を選択する．ついで，Take a New Snapshot をクリックする．例えば，Capture mode で Window Under Cursor を選択しておれば，マウスの形状が「＋」マークになり，キャプチャーするウィンドウをマウスでクリックする．その後，保存するファイル名を聞いてくるので，Save As を選択して適当なファイル名を入力する．



付録 5-図 1 KSnapshot の操作画面

2017 年 3 月 2 日更新

付録 6：カメラモジュールと OpenCV の利用

カメラモジュールの有効化

raspi-config を起動する．Interfacing Options から P1 Camera を選択する．カメラを有効化するかという問いに，<はい>と答え，raspi-config を終了する．Raspberry Pi を再起動すると，カメラモジュールが利用可能になる．

C/C++と OpenCV からカメラモジュールを利用する

C/C++用 OpenCV のインストール

```
$ sudo apt-get install libopencv-dev
```

で OpenCV をインストールする．このようにインストールすると，ヘッダファイルは /usr/include/opencv，/usr/include/opencv2 に保存される．opencv_core.a などのライブラリは/usr/lib/arm-linux-gnueabihf の下に保存される．そのため，OpenCV を使ったソースコードをコンパイルするためには，

```
$ g++ –o sample sample.cpp –I/usr/include/opencv –I/usr/include/opencv2 -lopencv_core –lopencv_highgui -lopencv_imgproc -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calid3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_flann –lpthread -ldl –lm
```

のような長いコマンドラインを入力する必要がある．このような場合，コマンドラインを記述したファイルを作り，それを

```
$ chmod 755 コマンドラインを記述したファイル名
```

のように実行可能にして利用するのが一つの方法である．また，Makefile を作り，make コマンドを利用する方法もある．なお，OpenCV は C から利用するバージョンを OpenCV1，C++から利用するバージョンを OpenCV2 として区別している．通常，OpenCV2 を C++から利用する．

UVC 対応のカメラから RasPi2 に画像を入力する

USB カメラの多くは UVC（USB Video Class）対応である．UVC 対応のカメラから OpenCV の機能を使って画像を入力し，ディスプレイに画像を表示するプログラムの例を付録 6-図 2 に示す．付録 6-図 2 の 7 行目では，USB カメラを cv::VideoCapture クラスの変数 capture として宣言する．capture の引数はカメラ番号であるが，-1 はデフォルトカメラを意味する値である．このプログラムを usbcam.cpp とすると，これをコンパイルするには，

```
$ g++ -o usbcam usbcam.cpp –I/usr/include/opencv2 –lopencv_core –lopencv_highgui
```

とする．

usbcam.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
#include <core/core.hpp>
#include <highgui/highgui.hpp>
#include <string>
#include <iostream>

int main() {
	cv::VideoCapture capture(-1);
	if (!capture.isOpened()) {
		std::cout << "Camera not found.\n";
		return 0;
	}

	capture.set(CV_CAP_PROP_FRAME_WIDTH, 640);
	capture.set(CV_CAP_PROP_FRAME_HEIGHT,480);
	cv::namedWindow("usb camera", CV_WINDOW_AUTOSIZE);

	while(1) {
		cv::Mat frame;
		capture >> frame;
		cv::imshow("usb camera", frame);

		if (cvWaitKey(1) == 'q') break;
	}

	return 0;
}
```

付録 6-図 2 UVC 対応のカメラから画像を入力し，それを表示するプログラムの例

Raspberry Pi 用カメラモジュールからの画像入力と OpenCV を使った処理

Raspberry Pi 用カメラモジュールの場合，付録 6-図 2 のように画像を取り込むことができない．raspicam_cv ライブラリを使う方法があるので，それを説明する．今回は，https://github.com/robidouille/robidouille/blob/master/raspicam_cv/README.md を参考にした．raspicam_cv ライブラリは https://github.com/robiouille/robidouille/tree/master/raspicam_cv から入手する．まず，cmake をインストールする．

```
$ sudo apt-get install cmake
```

次いで，raspberry pi userland ライブラリをインストールする．

```
$ mkdir ~/git
$ cd ~/git
$ mkdir raspberrypi
$ cd raspberrypi
$ git clone https://github.com/raspberrypi/userland.git
$ cd userland
$ ./buildme
```

ここで少し時間を要する．

```
$ cd ~/git
$ git clone https://github.com/robidouille/robidouille.git
```

```
$ cd robidouille/raspicam_cv
$ mkdir objs
$ make
```

インストール後，~/git/robidouille/raspicam_cv/にある libraspicamcv.a を/usr/lib にコピーする．

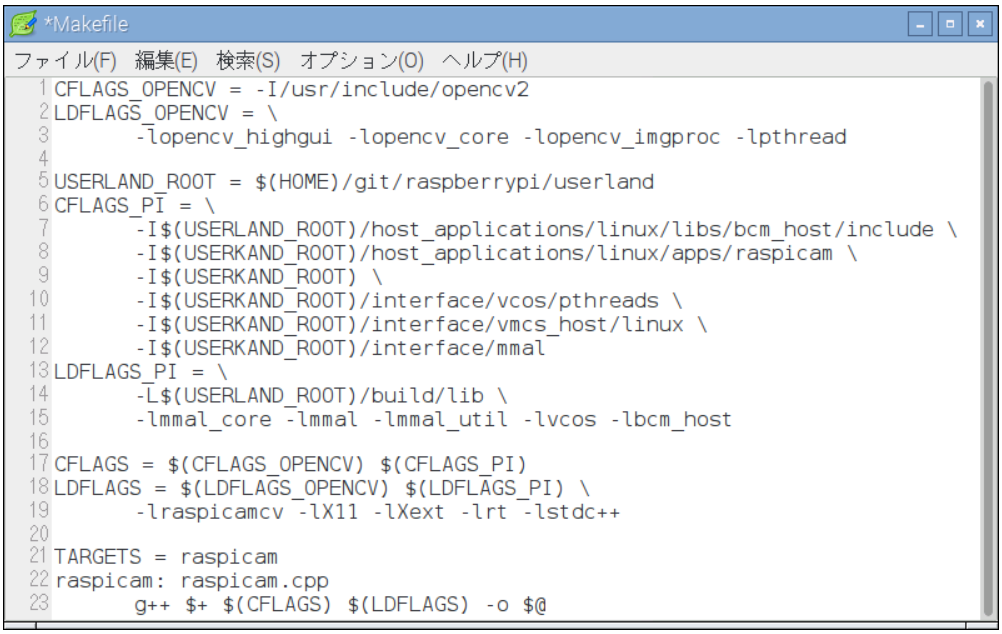
カメラモジュールから映像を入力し，それをディスプレイに表示するサンプルプログラムを付録 6-図 3 に示す．4 行目の”RaspiCamCV.h”は，~/git/robidouille/raspicam_cv/にある RaspiCamCV.h をカレントディレクトリにコピーした．25 行目で cv::Mat 型の変数 img にカメラモジュールの画像を取り込んでいる．そこまでに，raspicam_cv ライブラリの関数を使ってカメラモジュールを変数 capture に関連付けている．付録 6-図 4 にサンプルプログラムをコンパイル・リンクするための Makefile の例を示す．インクルードファイルとリンクファイルが多くあるので，注意が必要である．このサンプルプログラムを画像サイズ 640×480 で実行させた場合 25fps 程度のフレームレートであった．

*raspicam.cpp

ファイル(F)　編集(E)　検索(S)　オプション(O)　ヘルプ(H)

```
1  #include <core/core.hpp>
2  #include <highgui/highgui.hpp>
3  #include <imgproc/imgproc.hpp>
4  #include "RaspiCamCV.h"
5
6  int main(int argc, char *argv[]) {
7
8        //カメラモジュールを利用できるようにするための初期設定
9        RASPIVID_CONFIG *config = (RASPIVID_CONFIG*)malloc(sizeof(RASPIVID_CONFIG));
10       config->width = 640;
11       config->height = 480;
12       config->bitrate = 0;
13       config->framerate = 0;
14       config->monochrome = 0;
15       RaspiCamCvCapture *capture =
16             (RaspiCamCvCapture *) raspiCamCvCreateCameraCapture2(0,config);
17       free(config);
18
19       //画像を表示するためのウインドウを定義する
20       cv::namedWindow("capture", CV_WINDOW_AUTOSIZE);
21
22       cv::Mat img; //カメラモジュールの画像を保存するための変数
23       while (true) {
24             //カメラモジュールから画像を取り込む
25             img = raspiCamCvQueryFrame(capture);
26             //その画像をウインドウに表示する。
27             //但し，ディスプレイへの表示はcvWaitKey()関数を実行したときに行われる
28             cv::imshow("capture", img);
29
30             if (cvWaitKey(1) == 'q') {
31                   imwrite("sample.jpg",img);
32                   break;
33             }
34       }
35
36       //終了時の処理
37       cvDestroyWindow("capture");
38       raspiCamCvReleaseCapture(&capture);
39       return 0;
40 }
```

付録 6-図 3 raspicam_cv ライブラリと OpenCV を使ったサンプルプログラム



付録 6-図 4 raspicam_cv ライブラリと OpenCV を使うときの Makefile の例

2017 年 3 月 2 日更新

PS.

raspicam_cv ライブラリを Opencv3.1,Opencv3.2 で make すると，cvRound への参照が解決できないという主旨のエラーが発生する．それを解決する方法は

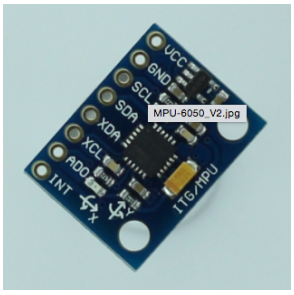
https://github.com/robidouille/robidouille/issues/16

に記述されている．2017 年 3 月 16 日

付録 7：加速度・角速度センサモジュール GY-521 の接続

センサモジュール，GY-521

GY-521 は，インベンセンス社の三軸ジャイロ・加速度センサ MPU-6050 を搭載したモジュールである．このセンサモジュールに関する情報は，インターネット上（例えば，「androcity MPU-6050 三軸加速度三軸ジャイロセンサーモジュール」など）から入手する．



付録 7-図 1 GY-521 の外観

RasPi2 との接続

GY-521 を，I2C インタフェースを介して RasPi2 と接続する．GY-521 の VCC 端子を RasPi2 の pin1（3.3V）に，GND を pin6（GND）に，SCL を pin5（GPIO3, I2C1 SCL）に，SDA を pin3（GPIO2, I2C1 SDA）に接続する．続いて，RasPi2 の I2C を有効にするために，次のように操作する．

① raspi-config の Interfacing Options で I2C を有効にする．
② ファイル /etc/modules の末尾に i2c-dev を加える．（デフォルトで設定されている模様）
③ sudo apt-get install i2c-tools libi2c-dev で，i2cdetect などのコマンドツールを使えるようにする．

リブート後，lsmod コマンドを入力すると i2c_dev と i2c_bcm2708 が表示される．sudo i2cdetect –y 1 を入力すると，接続されているデバイスのアドレスを確認することができる．GY-521 を接続している場合，0x68 に接続されていることがわかる．

2017 年 3 月 2 日更新

付録 8：無線 LAN への接続

USB タイプの無線 LAN アダプタを用いて、Raspberry Pi を無線 LAN に接続することが可能である。ただし、Raspberry Pi の USB ポートは供給できる電流が小さく、無線 LAN アダプタの機種によっては、動作が安定しないので注意が必要である。

無線 LAN アダプタを USB ポートに挿入する。自動的にドライバーがインストールされ、利用可能になる。このことは、lsusb コマンドで無線 LAN アダプタの製品名が見えることや、iwconfig コマンドで wlan0 が存在することなどで確認することができる。

続いて、/etc/network/interfaces というファイルの内容を、次のように書き換える。

```
interface wlan0 inet manual を interface wlan0 inet dhcp に
wpa-roam /etc/wpa_supplicant/wpa_supplcant.conf を
                    wpa-conf /etc/wpa_supplicant/wpa_supplcant.conf に
```

そして、/etc/wpa_supplicant/wpa_supplcant.conf を次のようにする。

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="KSU2010gx"
        proto=RSN
        key_mgmt=WPA-EAP
        pairwise=CCMP
        auth_alg=OPEN
        eap=PEAP
        identity="ユーザ ID"
        password="パスワード"
}
```

続いて、sudo ifdown wlan0 コマンドで無線 LAN を切り、sudo ifup wlan0 で無線 LAN を接続する。接続された旨の表示がされ、ping www.yahoo.co.jp などに対して正しい応答があれば接続されている。無線 LAN への接続は、必ずしも一発で接続できないので、その場合には、ifdown と ifup を繰り返す。

なお、wpa_supplcant.conf の書き方は、検索で見つけることができる。記述内容は、無線 LAN 環境によって異なる。上記の内容は、KSU2010gx に対する京都産業大学での設定を、該当する Web ページで確認して設定したものである。

市販の無線 LAN アダプタの中で、planex GW-USNANO2A は消費電力が小さく（1.3W）、Raspberry Pi に向いている。他の製品は 2.5W 程度のものが多い。

Raspberry Pi には、Wi-Fi Config というツールが用意されており、上記の手順をほぼ自動的にやってくれる。ただし、このツールが全ての WiFi 環境に対応しているわけではないので、設定ファイルを直接操作できることが望ましい。

2016 年以前の情報

付録 9：Wiring Pi のインストール

Wiring Pi は Gordon Henderson 氏が開発した Raspberry Pi 用 GPIO のライブラリで，GPIO を利用するプログラムを C 言語で作成する際に役立つ（http://wiringpi.com）.

WiringPi を使用するには libi2c-dev という I²C の開発用ライブラリが必要である．それを次のようにインストールする.

```
$ sudo apt-get install libi2c-dev
```

WiringPi はソースコードをダウンロードしてビルドすることによってインストールする．Git というバージョン管理システムを使ってソースファイルを入手する．Git を次のようにインストールする.

```
$ sudo apt-get install git-core
```

次に WiringPi のソースファイルをダウンロードする.

```
$cd ~
$ git clone git://git.drogon.net/wiringPi
```

ソースファイルを次のようにコンパイルする.

```
$ cd ~/wiringPi
$ ./build
```

正常にコンパイルが完了すると，ライブラリや gpio ユーティリティがインストールされる．次のコマンドで gpio ユーティリティのバージョンを確認できる.

```
$ gpio –v
```

注）上の手順の中で，git clone …でファイルをダウンロードできない場合（京産大の学内ネットワークで，情報センターに登録されていない Raspberry Pi ではできなかった），他のパソコンやネットワークを通じてソースファイルをダウンロードし，それを RPi2 にコピーする.

2017 年 3 月 2 日更新

付録 10：LeafPad，nano，vi のタブキーの設定変更

Python や C/C++のプログラムを作成するときに，タブキーが半角 8 文字分になっているとソースコードが横に長くなりすぎる．半角 4 文字か半角 2 文字が適当であろう．Raspberry Pi で用いるテキストエディタ Leafpad, nano, vi でタブキーの設定は，次のように変更する．

Q: Leafpad のタブをデフォルト（半角 8 文字）から半角 4 文字に変更する．
A: 設定ファイル/usr/share/raspi-ui-overrides/applications/leafpad.desktop の 43 行目を Exec=leafpad %f から Exec=leafpad --tab-width=4 %f に変更する．

Q: nano のタブをデフォルト（半角 8 文字）から半角 4 文字に変更する．
A: 設定ファイル~/.nanorc に set tabsize 4 を記述する．

Q: vim の設定ファイル.vimrc に設定方法．
A: vim の初期設定を行うファイル~/.vimrc を次のように記述すると良い．
set tabstop=4
set showmode
set number
set nocompatible
第一行はタブを半角 4 文字にする（デフォルトは 8 文字）．第 2 行は現在のモードを表示する．第 3 行は行番号を表示する．第 4 行は vi コンパチブルを解除する．vi コンパチを解除すると，入力モードで矢印キーによるカーソル移動が可能になる．

2017 年 3 月 2 日更新

付録 11：MacBoook から無線 Lan で Raspberry Pi をリモート操作する方法

1. MacBook を無線 Lan のアクセスポイントとして設定する

MacBook の無線 Lan は、通常、アクセスポイント（AP）に接続するための子機として利用する。今回は Raspberry Pi の無線 Lan に対する AP として設定する。

MacBook を USB Ethernet アダプタを使って、LAN に接続しておく。「システム環境設定」を起動し、「共有」を選択する。共有パネルの左側にあるサービスから「インターネット共有」をクリックする（この時点では、入にチェックを入れないで、選択するだけ）。右側の「共有する接続経路」は「USB Ethernet」を選ぶ。「相手のコンピュータが使用するポート」は「Wi-Fi」を選ぶ。

右下にある「Wi-Fi オプション」をクリックすると、下のようなパネルが表示される。ネットワーク名とパスワードは各自で適当なものを入力する。チャネルとセキュリティは図のようにする。「OK」をクリックして戻る。

図 1 Wi-Fi オプションの設定画面

共有パネルのサービスで「インターネット共有」の入にチェックを入れる。確認のダイアログボックスが表示されるので「開始」をクリックする。「インターネット共有」のインジケータが緑になり、設定が有効になったことがわかる。また、メニューバーにある WiFi の扇型アイコンの中に「↑」が現れる。「システム環境設定」を閉じる。

2. Raspberry Pi の無線 Lan 設定

USB タイプの無線 Lan アダプタを接続する（Planex の GW-USNANO2A は消費電力が小さく（1.3W）、Raspberry Pi に向いている）。初めてアダプタを接続すると、ドライバーが自動的にインストールされる。lsusb コマンドや iwconfig コマンドを入力し、wlan0 が存在す

ることを確認する。wlan0 は 1 台目の無線 Lan を示す名前である。

sudo iwlist wlan0 scan | grep ESSID を入力し、図 1 で設定したネットワーク名が ESSID の一つとして表示されていることを確認する。設定したネットワーク名が表示されない場合、MacBook のメニューバーに、WiFi アイコンが「↑」付きで表示されているか？など、1.で行った設定を確認すること。

/etc/network/interfaces ファイルを次のように設定する。

```
auto lo
iface lo inet loopback
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wap_supplicant.conf
iface default inet dhcp
wireless-power off
```

/etc/ wpa_supplicant/wap_supplicant.conf ファイルを次のように設定する。

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid=”設定したネットワーク名”
        psk=”設定したパスワード”
        key_mgmt=WPA-PSK
}
```

sudo ifdown wlan0 で無線 Lan を切断し、sudo ifup wlan0 で無線 Lan を接続する。リブートすれば、自動的に無線 Lan に接続する。ifconfig コマンドなどを使って、IP アドレスを確認する。

3. xrdp リモートデスクトップ

ネットワーク接続されたコンピュータから、他のコンピュータを操作するデスクトップ環境をリモートデスクトップとよぶ。VNC や xrdp などのソフトウエアがある。ここでは、xrdp を使う。次の手順で、Raspberry pi に xrdp をインストールする。

```
sudo apt-get update
sudo apt-get install xrdp
```

日本語キーボードの配列を指定するために、次の操作を行う。

```
cd /etc/xrdp
sudo wget http://w.vmeta.jp/temp/km-0411.ini
sudo ln -s km-0411.ini km-e0010411.ini
sudo ln -s km-0411.ini km-e0200411.ini
sudo ln -s km-0411.ini km-e0210411.ini
sudo service xrdp restart
```

Mac 用の xrdp クライアント（Microsoft Remote Desktop）がマイクロソフトから提供されている。

```
https://itunes.apple.com/jp/app/microsoft-remote-desktop/id715768417?mt=12
```

の手順にしたがってインストールする。

Microsoft Remote Desktop を開始する。図 2 のような画面が表示される（ここで Raspi2 となっている部分は、最初はない）。

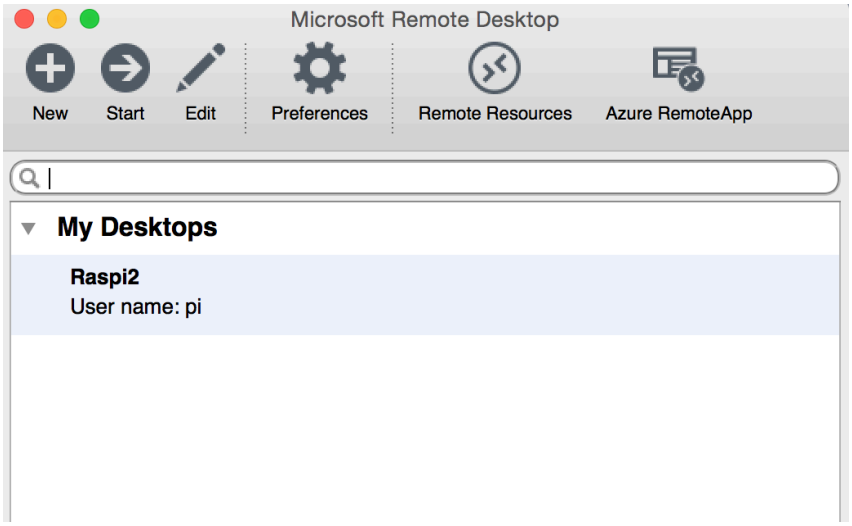


図 2 Microsoft Remote Desktop の画面

New をクリックすると、図 3 のような表示になる。Connection name は好きな名前を入力する。PC name は、2.の最後に確認した Raspberry Pi の wlan0 の IP アドレスを入力する。User name と Password は Raspberry Pi 側のもので、初期状態では pi と raspberry である。Resolution と Colors は自分の MacBook の仕様に合わせる。その他は、そのままで良い。この画面を左上の●をクリックして閉じる。

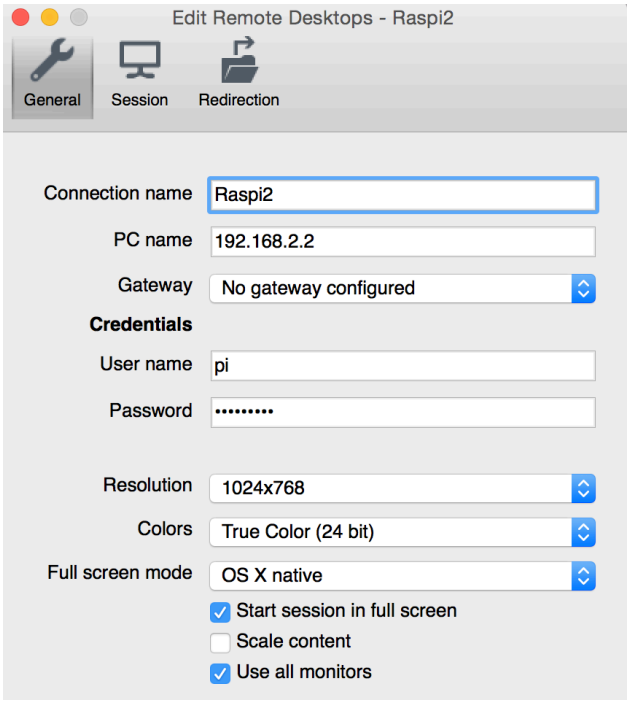


図 3 Edit Remote Desktop の画面

元の画面に戻ると、設定した Connection name が表示されている。これをダブルクリック（あるいは選択して Start）する。全ての設定が正しければ、Raspberry Pi のデスクトップが表示され、操作可能になる。一旦、操作可能になれば、Raspberry Pi 本体側のケーブル類は、電源を除いて取り除くことができる。

全ての設定を正常に終了し、MacBook から無線 LAN で Raspberry Pi を使う場合には次のようにする。

(1) MacBook を USB Ethernet で LAN に接続して起動する。メニューバーにある WiFi の扇型アイコンの中に「↑」が現れていることを確認する。

(2) Raspberry Pi を起動する。ブートアップの完了を待つ。

(3) MacBook で xrdp を起動し、Raspberry Pi に接続する。

2016 年以前の情報

付録 12：7 インチ公式タッチパネルディスプレイの接続

Raspberry Pi 7 インチ タッチ・スクリーンディスプレイの接続・取り付けを解説している Web サイト（例えば，https://raspberry-pi.ksyic.com/page/page/pgp.id/4）を参考に接続する．これだけで正しく表示されない場合，次のことを確認する．

(1) /boot/config.txt の display_default_lcd=0 の行を#でコメントアウトする

(2) 上下逆さまに出力するときは，さらに lcd_rotate=2 を書き加えると，正常に出力する．

2016 年以前の情報

2017 年春学期に使う RasPi2 の状態

1. NOOBS_v2_3_0
2. 付録 1 に記した日本語環境
3. 付録 2 に記した emacs
4. 付録 5 に記した KSnapshot
5. 付録 6 に記したカメラモジュール関係環境（OpenCV, raspicam_cv ライブラリを含む）
6. 付録 7 に記した I2C の環境
7. 付録 9 に記した WiringPi