Raspberry Pi Mouse

取扱い説明書

3.0 版
2020 年 3 月 1 日
株式会社アールティ

RT CORPORATION

内容

RT CORPORATION

1 安全について

1.1 安全にご使用いただくために

① Raspberry Pi Mouse をお買い上げいただきましてありがとうございます。ご使用になる前にこの説明書をよくお読みになり、十分理解した上で作業を始めてください。

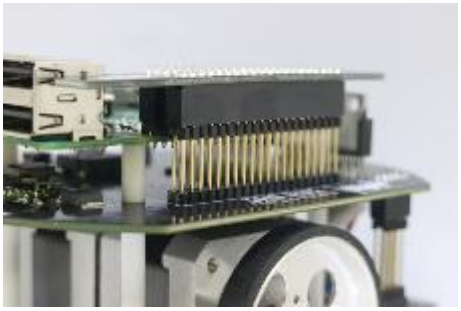
② 初めてロボットを触る方は、経験者と一緒に作業することをお勧めします。

③ 【注意】ショートについて

センサ、モータ、基板、電池などの端子同士など接触によるショート時にはロボットを壊すだけではなく、発火の恐れがあります。ケーブルの取り回しの際には各部品に挟まれないよう余裕をもたせ、ケーブルに傷がある場合は絶対に使用しないでください。また、電源とグラウンド線の扱いには細心の注意を払ってください。

④ 電源の投入について

電源を入れる前にもう一度 RaspberryPi と本体の接続が正しいことをご確認して電源を入れてください。



正しい接続例

間違った接続例

⑤ 【重要】電池に関するご注意

Raspberry Pi Mouse は、リチウムイオンポリマ充電池(以下Li-Po電池と表記)を採用してい

ます。Li-Po電池を使用するにあたり、必ず正しい知識で取扱いをお願いします。Li-Po電池は、小さくて軽く、瞬間的に流せる電流もほかの電池に比べて大きく、さらにメモリ効果もほとんどないためロボット用途には向いています。しかし、他の電池に比べるとやや高価です。過充電やショートを起こすと発火・爆発することもあります。一般的なLi-Po電池は保護回路が入っていますが、取扱いには細心の注意が必要です。Li-Po電池はそのCellの数で電圧の大きさが決まります。1Cell平均3.7V、2Cellで7.4V、3Cellで11.1Vと電圧が大きくなっていきます。また、全ての電池の放電に関する単位にC を使います。C は容量に対する放電の比率を表し、1C ですと1 倍、2C ですと2 倍、3C ですと3 倍の電流を放電するという意味になります。
充電する際には専用の充電器を使ってください。(Li-Po電池は1Cで充電するのが良いでしょう)保管する際には充電容量の約80%(約12.0V)の状態で保管するのが良いと言われています。また、1Cell 3.3V を切ると過放電となり、使えなくなってしまいます。Raspberry Pi Mouseで使われているLi-Po電池は3Cell なので、10V 以下には絶対しないでください。目安としてモータに電源が入っている場合、フル充電から20分ぐらいで10V程度まで電圧が下がります。**10V以下になるとピー・ピーと警告音のブザがなります。ブザがなりましたら、早急に電源をOFFにしてください。**Li-Po電池を利用するときは、注意事項をよく守って、過放電、過充電にならないように取り扱ってください。
万が一、誤った使用により怪我をしたり、火災を起こしたりした場合でもメーカ販売店は責任を持ちません。
出荷時、電池は100%の充電はされていません。ご使用前に充電してからご使用ください。

充電について

必ず専用充電器を使用してください。充電中は燃えやすいものが近くにない場所で、目を離さないように行って下さい。充電完了後は、必ず電池のコネクタを充電器からはずしてください。またLi-Po電池はメモリ効果がほとんどないため、追加充電可能です。必ず完全に放電してしまう前に充電してください。

＜充電中に電池がふくれ始めたら＞
万一、充電中に異臭やふくれ始めたのを確認したら、直ちに充電を中止し、コネクタを充電器からはずしてください。（充電し続けると発火、または爆発の危険があります。）その場合、電池を外して燃え移るものがない安全な場所で1 日程度様子をみてください。再使用は絶対にしないで廃棄してください。（廃棄方法の項を参照して廃棄処分してください。）

放電について

Raspberry Pi Mouseに使用しているLi-Po電池は過放電をすると電池自体が使用不可能になります。過放電は絶対にしないよう注意してください。また、Raspberry Pi Mouseをご使用後は**必ずコネクタを抜いてください**。

使用について

本説明書をよくお読みになり、正しくご利用ください。ショート、衝撃、釘刺し等による破損でも発火、爆発の危険があります。たとえば、電池と鋭利な工具とを一緒に工具箱に入れる、汗をかいた手でコネクタを触る、誤って水に落とすなど、いろいろな場面が想定されますが、そのような行為は、絶対にしないでください。
Li-Po電池は、定格電流が決められています。Raspberry Pi Mouseのみでご使用の場合では最大電流を超えないように設計していますが、各ユニットパーツを自作等のものに取り替え

たときやマニュアルに無いような応用回路にしたとき、電池の放電容量を超えないように設計してください。定格電流以上に電流を流すと、爆発の可能性があります。

保管方法について
保管する際十分な充電を行い、周りに導通するものや、燃えやすいものがないところに保管してください。

廃棄方法について
海水と同程度の濃度の塩水に2～3 日漬けて完全に放電させてから、燃えないごみとして廃棄して下さい。具体的には水1L に対して塩30g を入れると海水と同程度の塩水になります。

Li-Po電池の使用における保証について
Li-Po電池を安全に使用するのはユーザの責任です。メーカおよび販売店は、Li-Po電池の誤使用によって起こるいかなる対人・対物事故、損害、破損について一切の責任を負いません。安全に管理すれば非常に使いやすい電池です。正しい知識をもってお取り扱いください。

2 ハードウェアについて

ここでは、Raspberry Pi Mouseの内容物と各部の名称を確認します。

2.1 パーツ一覧

組み立てられた状態で出荷されますので、パーツとしては本体、Raspberry Piと本体をつなぐコネクタ、Raspberry Pi(フルキットのみ)、Debian系OSのRaspbian 入りMicroSDカード(フルキットのみ)になります。

2.2 名称について

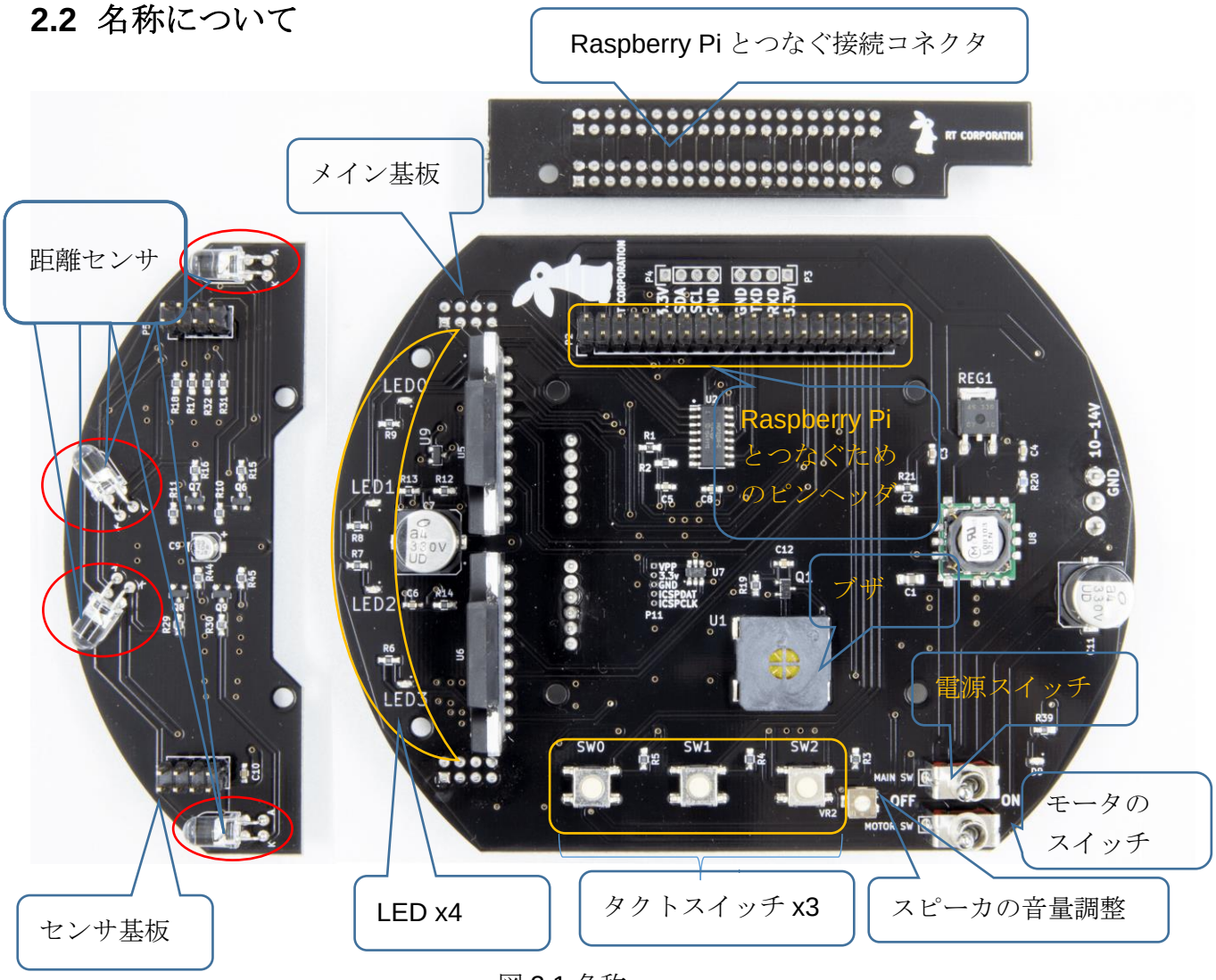
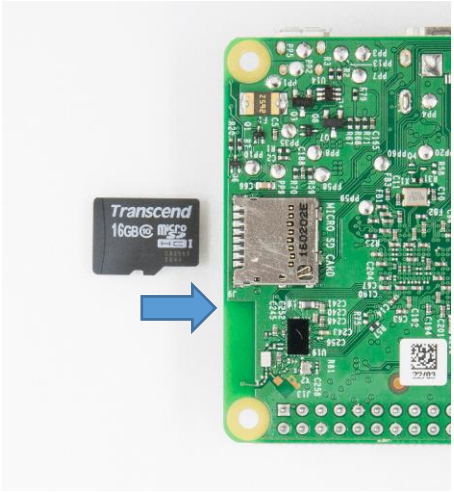


図 2.1 名称

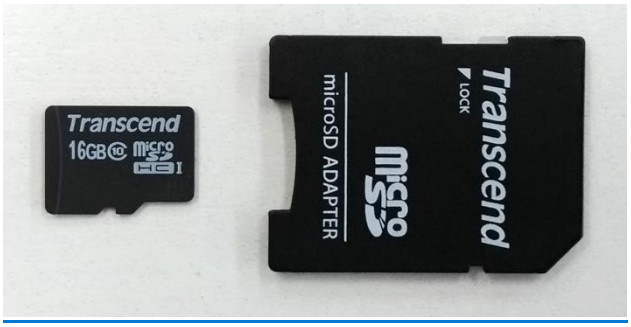
3 Raspberry Pi Mouse の取り扱い

3.1 本体の組み立て

OS が入った microSD カードを図の向きで挿入します。Raspberry Pi3 では、ロック機能がなくなっています。

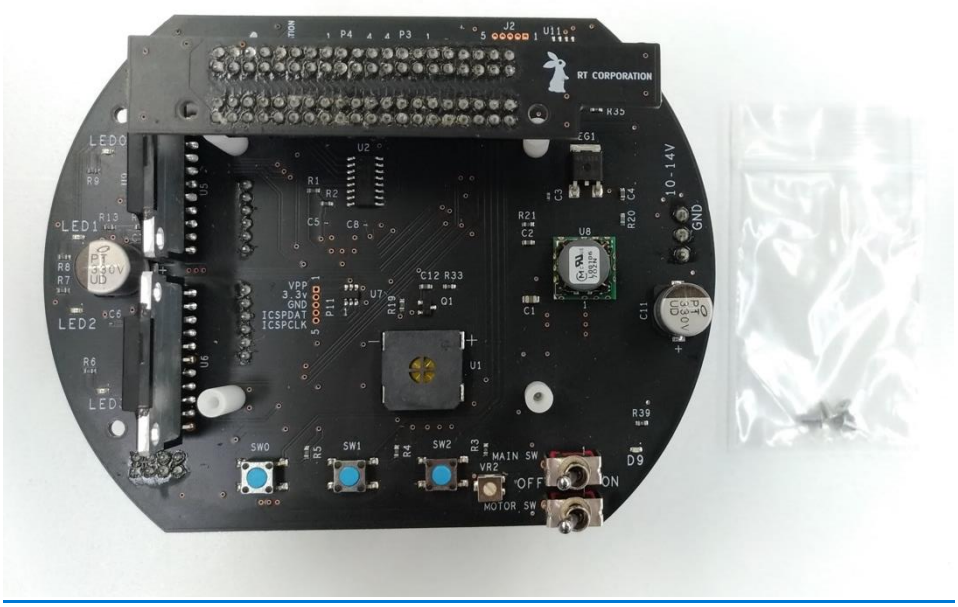


RaspberryPiMouse フルキットを購入された方は、図のように microSD カードを SD カードに変換するアダプタから取り外した後、RaspberryPi に挿入してください。

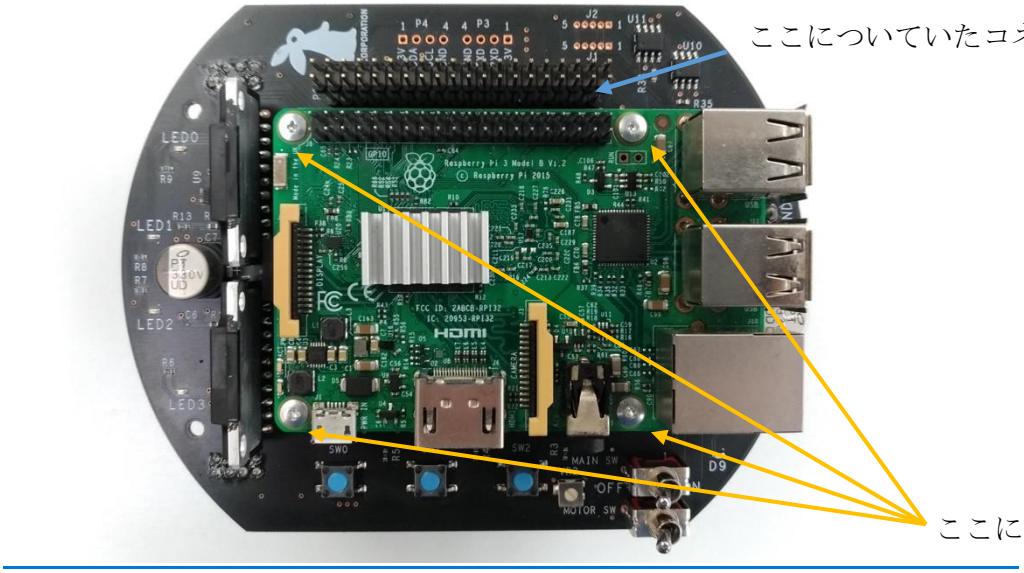


Raspberry Pi をつなぐコネクタと Raspberry Pi を固定するネジを外し、スペーサの上に Raspberry Pi を置きます。

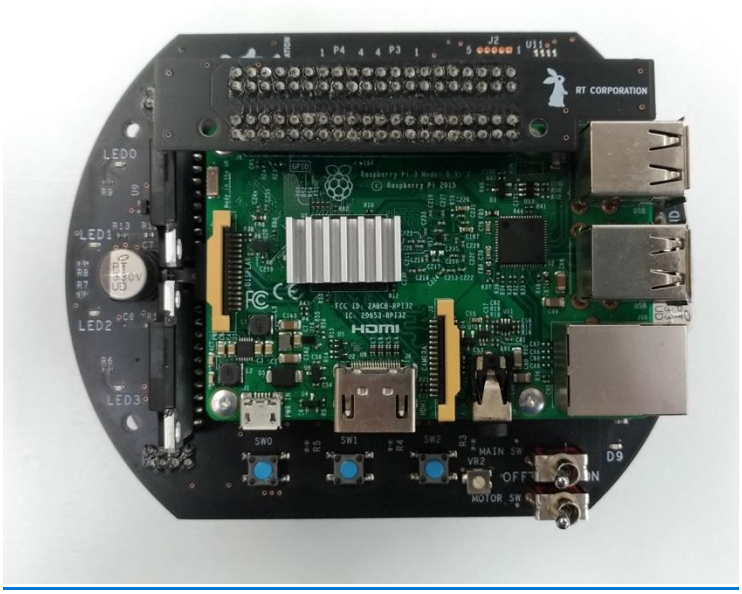
Raspberry Pi Mouse と RasberryPi 取り付け用のネジを用意します。



スペーサーの上に RaspberryPi を置き、RaspberryPi を置いたら、同梱されているネジで固定してください。

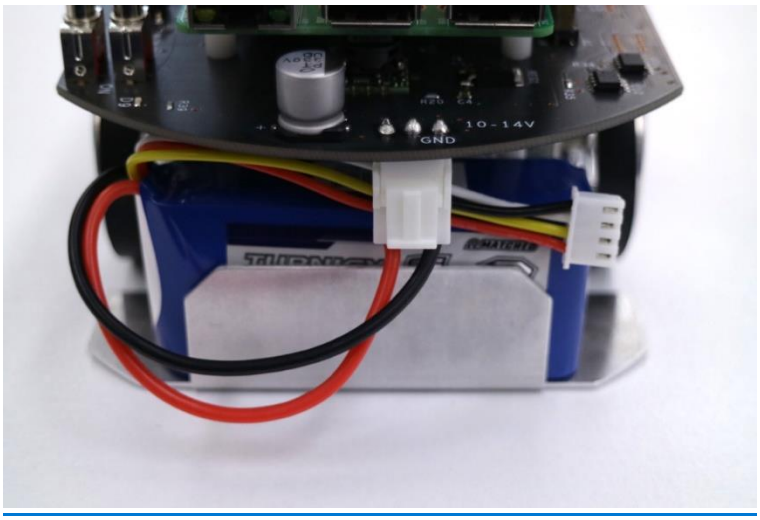


コネクタで Raspberry Pi と Raspberry Pi Mouse を接続します。ピンがずれないように入れてください。



3.2 電池の接続

電池は、Raspberry Pi Mouse の後方に図のように搭載して接続します。電池を接続する際は、電池の容量が十分に残っていることを確認し、「電池に関するご注意」をよく読んだ上でご使用ください。



電池の充電方法の一例として、Raspberry Pi Mouse フルキットに付属の充電器を用いた方法を紹介します。まず AC アダプタと充電器をように接続します。

電池と充電器を図ように接続します。コネクタの向きを間違えないよう注意してください。

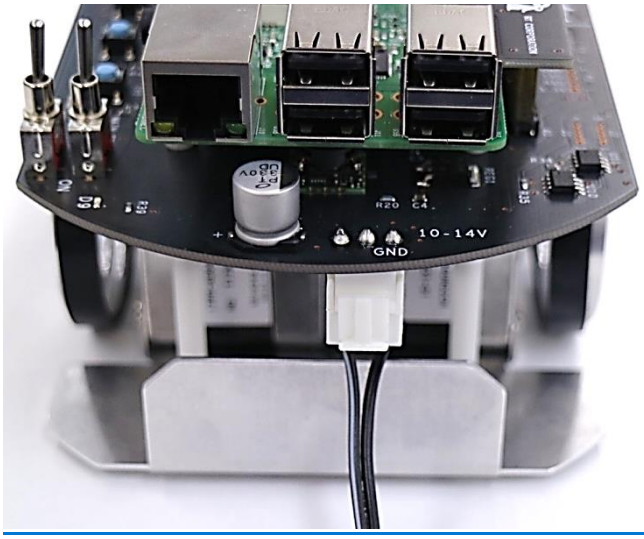
充電中は、充電器の赤い LED が点灯します。

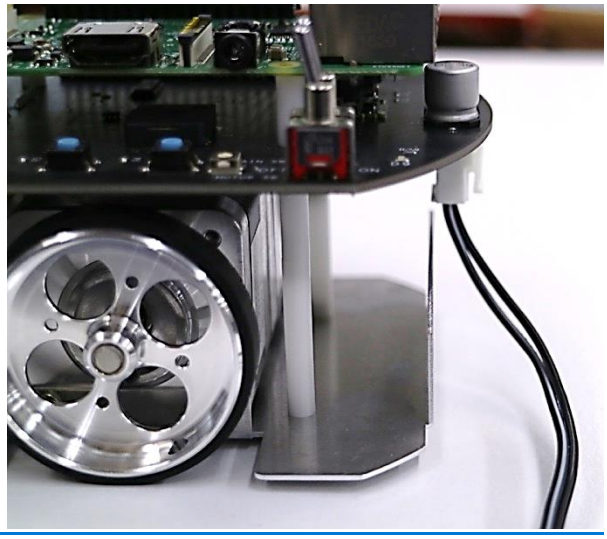
充電が終了すると、充電器の緑色の LED が点灯します。終電が終了したら、電池を充電器から取り外してください。

3.3 AC アダプタの接続

Raspberry Pi Mouse フルキットをご使用の方は、電池の代わりに AC アダプタをご使用いただくこともできます。AC アダプタをご使用になる場合は、まず付属の変換ケーブルと AC アダプタを図のように接続します。

変換ケーブルを介して、図のように接続します。接続する際は、コネクタの向きに注意してください。

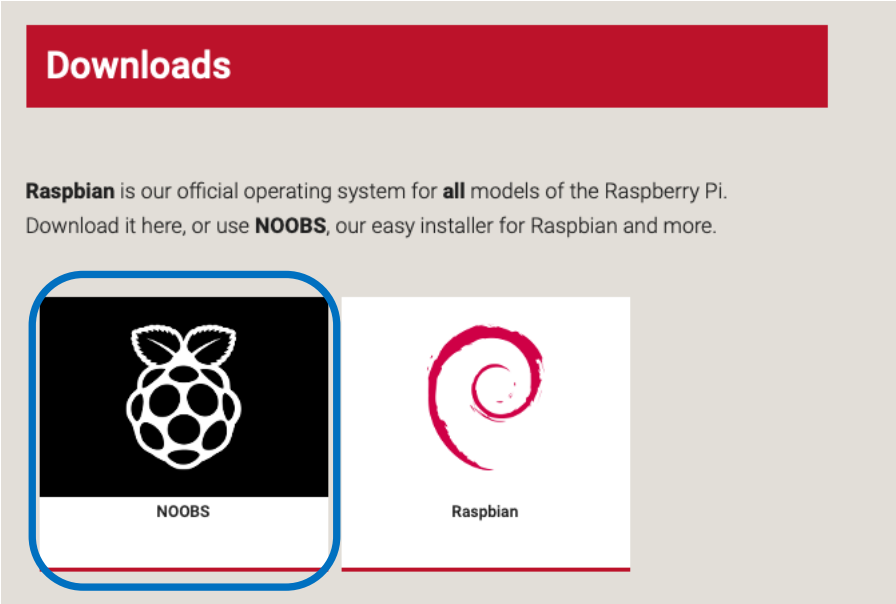




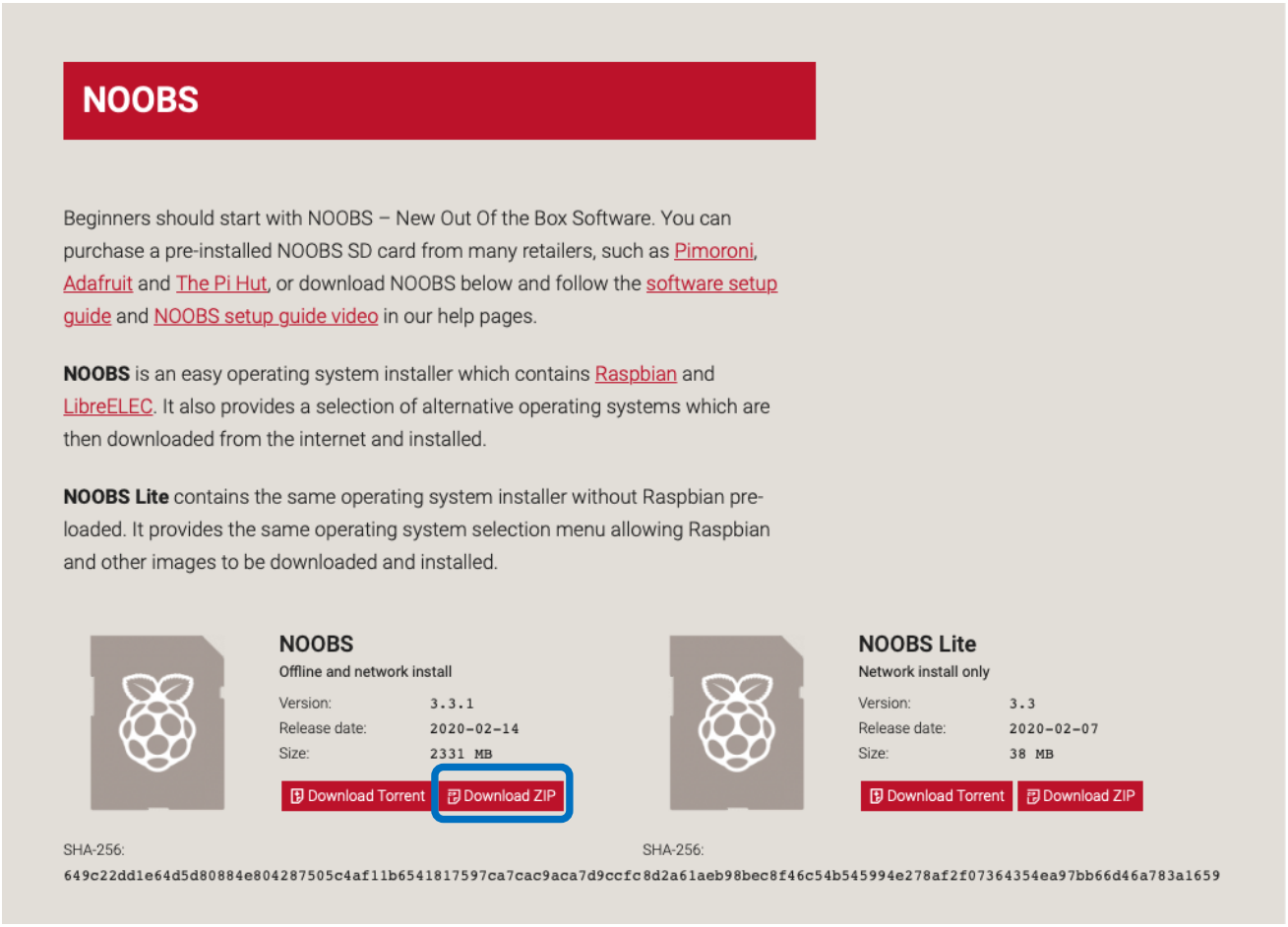
4 OS のインストール方法について

Raspberry Pi は Raspbian、Ubuntu(ROS)、RTM にて動作確認ができていますが、ここでは Raspberry Pi 公式の OS Raspbian のインストールを行う手順を示します。Raspbian の OS は
https://www.raspberrypi.org/downloads/
からダウンロードして SD カードに書き込む必要があります。

Windows 上での作業方法を掲載します。

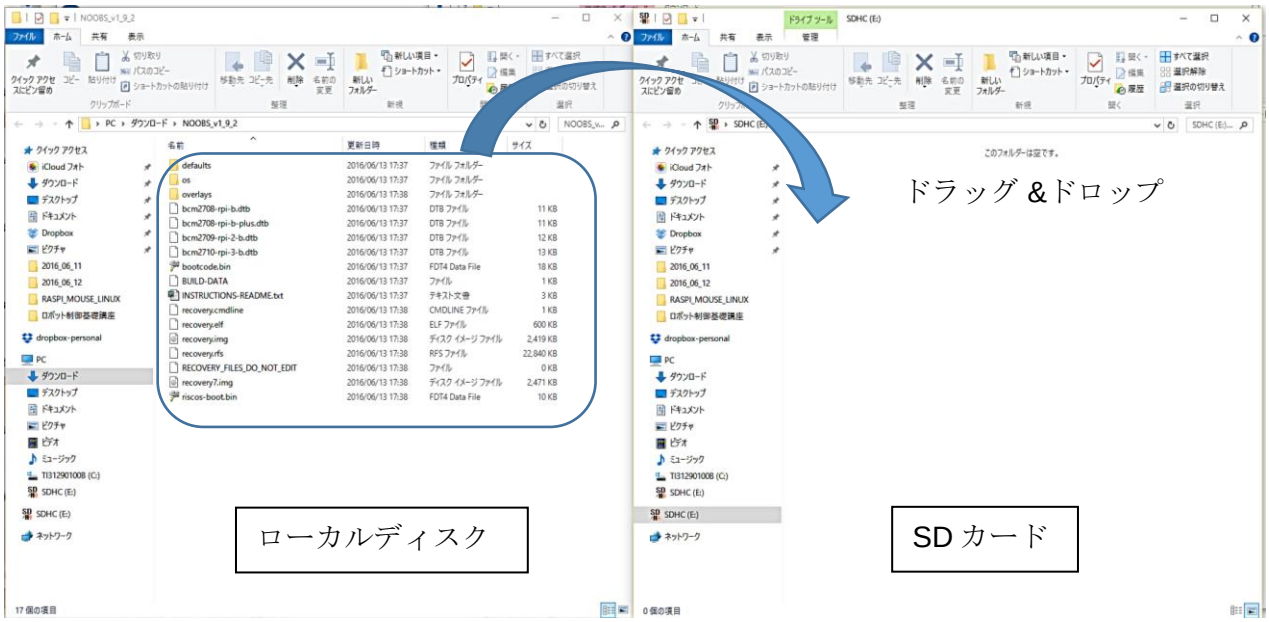


ダウンロードサイトには上記の 2 種類(NOOBS と Raspbian)があります。Raspbian の方はイメージファイルになります。ダウンロードして SD カードに書くにはイメージを書くソフトが必要になります。NOOBS は、ダウンロードし、展開したデータをそのまま SD カードにコピーするだけです。初心者には NOOBS をお勧めします。ここでは、NOOBS での Raspbian のインストール方法を示します。NOOBS をクリックしていただくと下記の画面になります。

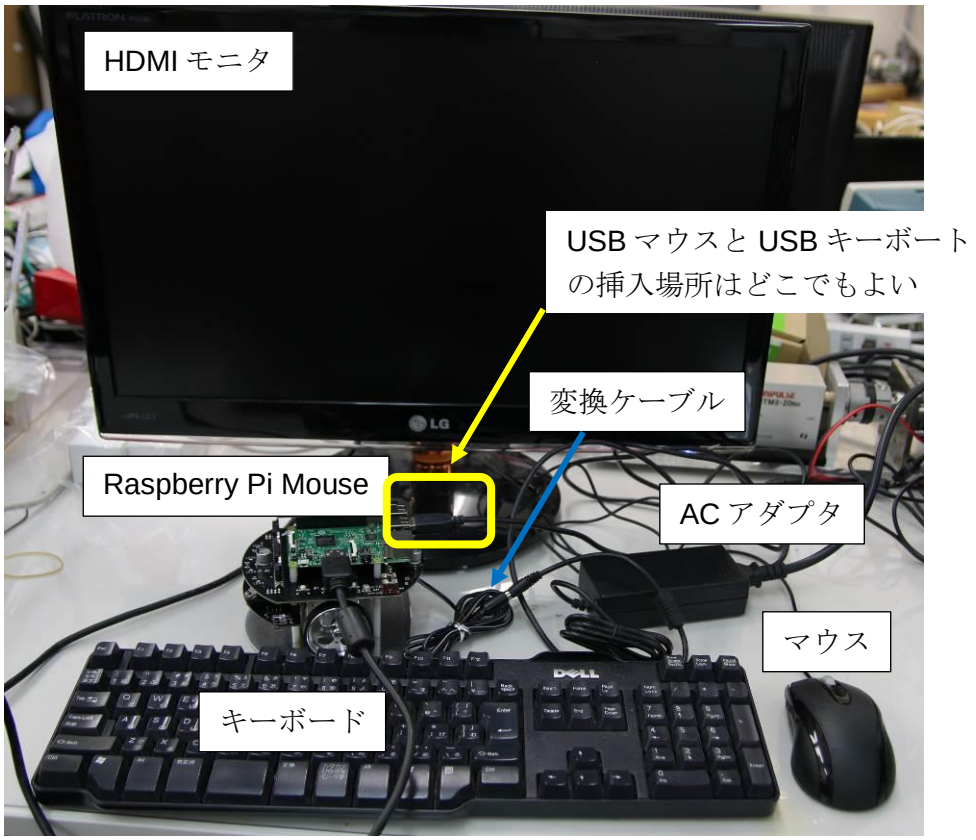


NOOBS(NOOBS LITE ではない方)の Download.ZIP をクリックしローカルに保存します。

　この ZIP を展開(windows7 以降なら右クリックし”すべて展開”で展開できます。)し、SD カードにコピーします。SD カードは 16GB 以上をお勧めします。デバイスドライバを Raspberry Pi 上でコンパイルする環境を整備すると 8GB 必要です。



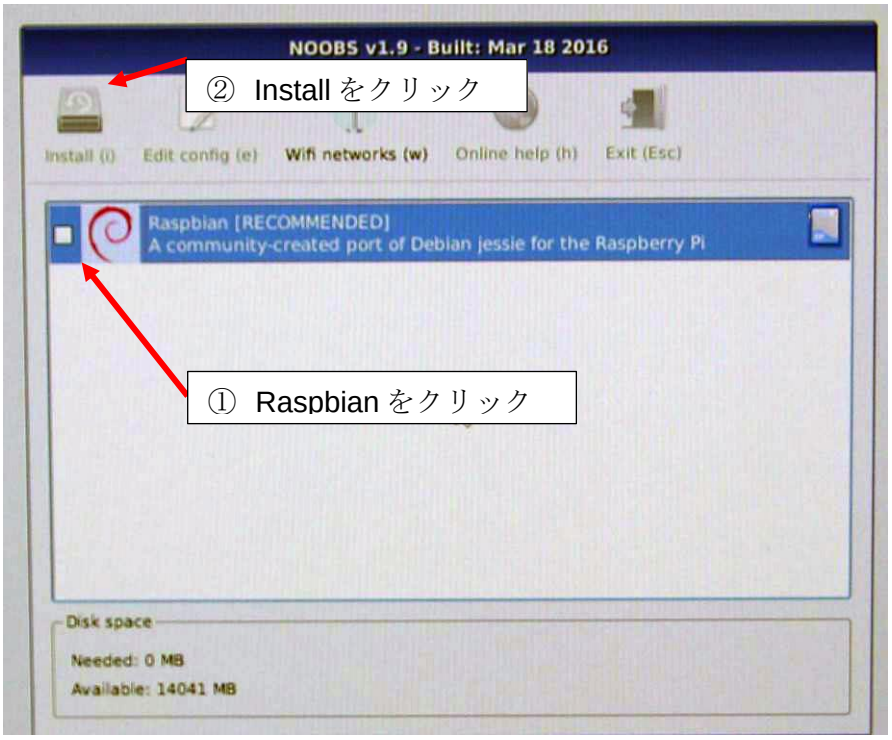
SD カードに OS の基となるデータを書き終えたら、3 章に示したように Raspberry Pi Mouse を組み立てます。組み立てましたら、Raspberry Pi に USB のマウスと USB のキーボードと HDMI のモニタを接続します。



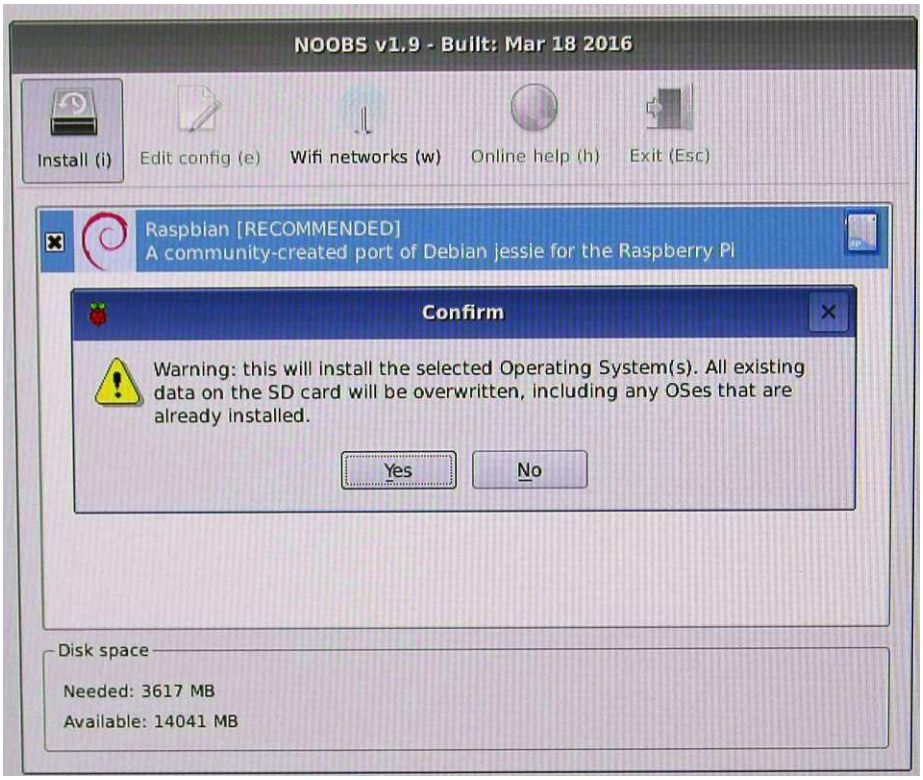
上記の接続例では、電池の代わりに AC アダプタで接続しています。(フルキットに含まれています。Raspberry Pi Mouse 単品で購入し AC アダプタがほしい方は、オプションで別途購入できます。(LiPo 充電器 LBC-010(AC アダプタ・変換ケーブル付き)7500 円(税抜))

電池で初期環境を作る際はフル充電した電池をご使用ください。マウスとキーボードと HDMI のモニタを接続しましたら、MAIN SW を ON にしてください。MOTOR SW は OFF にしてください。OS が起動する前で MOTOR SW を ON にするとモータに電流が流れます。電池で環境整備をしている場合、環境の設定が終わる前に電圧が 10V 以下になる可能性があるため、環境整備の時は必ず、MOTOR SW は OFF にしてください。

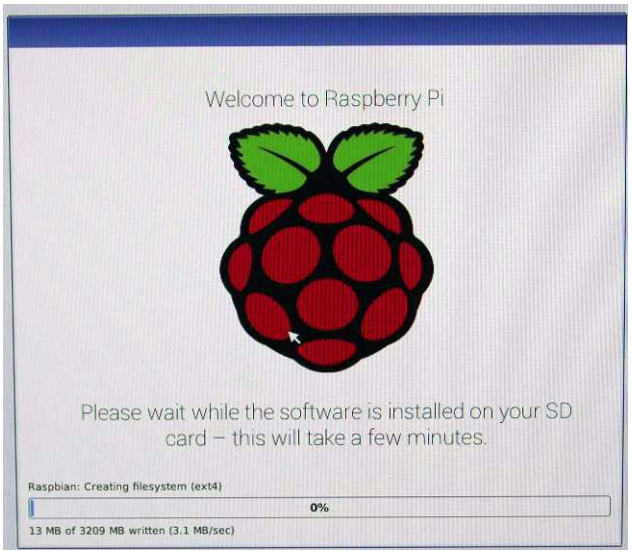
デバイスドライバは github からダウンロードしますので、LAN ケーブルを挿入するか無線 LAN で接続できるようにしておいてください。



MAIN SW を ON にし、しばらくすると上記の画面が表示されます。Raspbian をクリックし、Install ボタンをクリックします。Raspbian をクリックしないと Install ボタンが押せません。



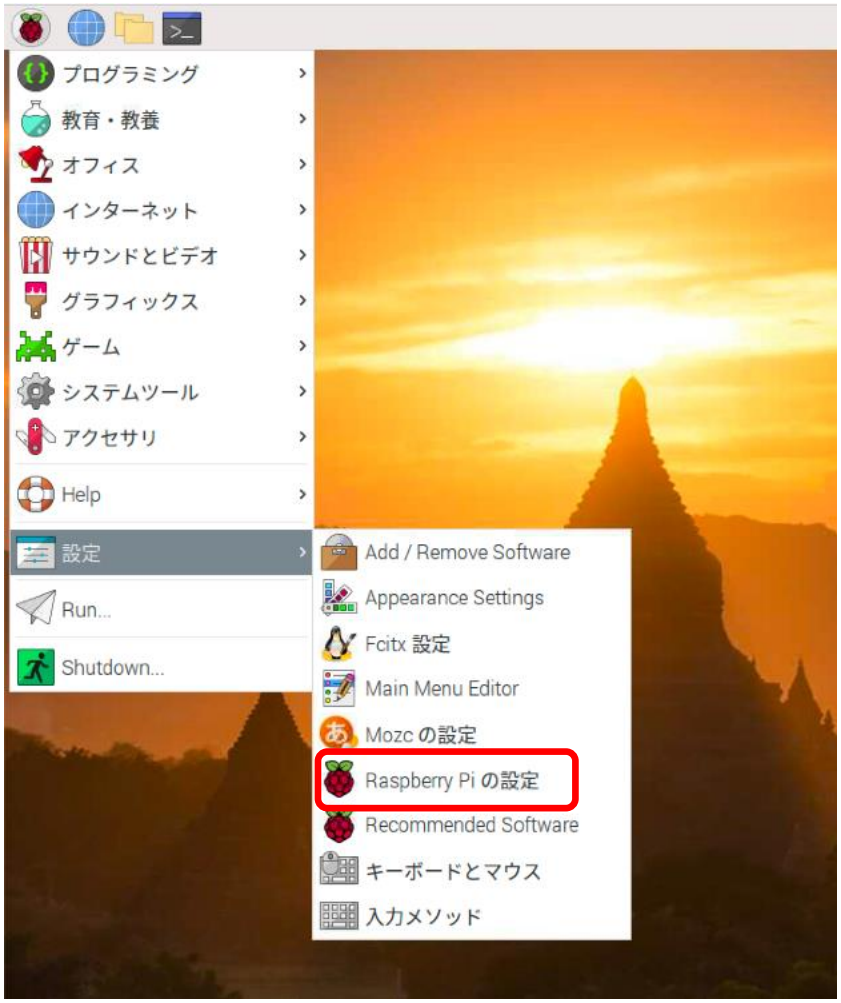
Install ボタンを押すと確認の画面が表示されますので”Yes”をクリックしてください。



“Yes”を押すとインストールが始まりますのでしばらくお待ちください。Raspberry Pi3 で 15 分ぐらいです。

インストールが終わると上記が表示されますので、”OK”ボタンをクリックします。クリックすると自動的に再起動します。

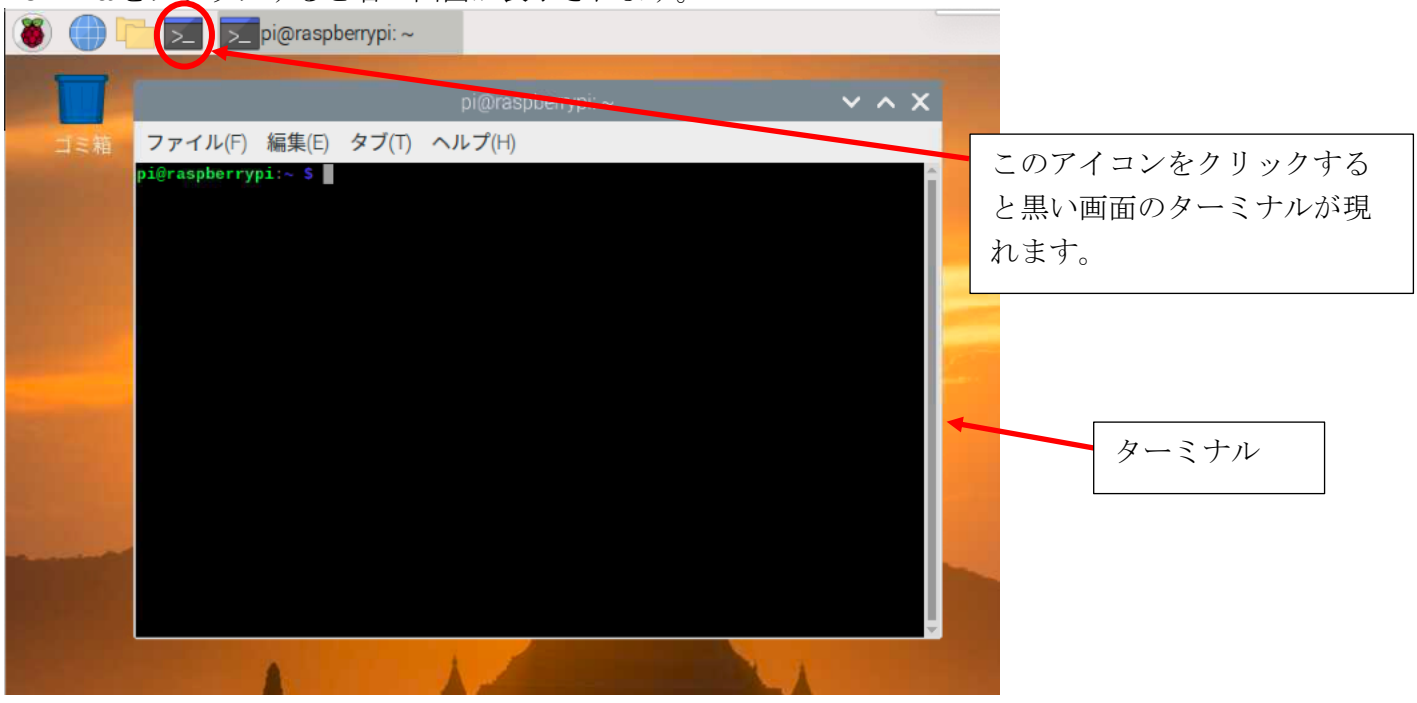
再起動すると GUI が立ち上がります。パスワードや user の問い合わせはありません。
Raspberry Pi Mouse は SPI、I2C を使用するため SPI、I2C の機能をイネーブルにします。
Menu の設定の Raspberry Pi の設定をクリックします。



Raspberry Pi の設定が立ち上がりますので、インターフェイスのタブをクリックします。SPI、I2C の有効をクリックし、”OK”ボタンをクリックします。
OS 入りの SD カードには、ここまで設定されたものが入っています。環境整備が面倒な方は Raspberry Pi Mouse フルキットをお勧めします。

5 デバイスドライバのインストールについて

デバイスドライバはGitを使って「GitHub」からダウンロードします。Gitはターミナルにコマンドを打ち込んで使用します。GUIからターミナルの起動は以下のようにして起動します。Terminalをクリックすると右の画面が表示されます。

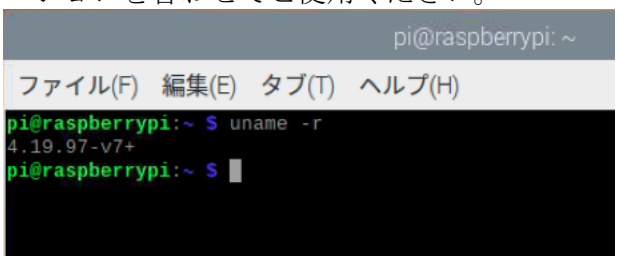


デバイスドライバのインストールはSSHでログインしても可能です。SSHでログインするには、初期ユーザ名 pi とパスワード raspberryを入力してください。

RT CORPORATION

SSHでログインするには、Raspberry Piの設定のインターフェイスにあるSSHを有効にする必要があります。

デバイスドライバは、GitHubからダウンロードしたファイルの「./lib/RaspberryPiのバージョン/カーネルのバージョン」にあるので、「uname –r 」でカーネルのバージョンを調べ、カーネルのバージョンを合わせてご使用ください。



Gitのクローンをします。(もしRaspberryPiMouseのディレクトリが既にある場合削除しておく)

```
git clone https://github.com/rt-net/RaspberryPiMouse.git
```

Gitからパスワードを聞かれた場合は、raspberryと入力してください。予め「sudo su -」でrootになるとパスワードが聞かれなくなります。

クローンすると以下のファイル構造が存在していると思います。

```
RaspberryPiMouse
 ├50-rtmouse.rules
 ├LICENSE
 ├README.md
 ├SampleProgram
 |  ├README,md
 |  ├Remocon_Step1.sh
 |  └など
 |
 ├cad_data
 ├lib
 |  ├Pi1B+
 |  |  └3.18.14+など
 |  |     └rtmouse.ko
 |  └Pi2B+
 |     └4.4.13-v7+など
 |        └rtmouse.ko
 ├src
 |  └drivers
 |     ├Makefile
 |     ├Makefile.rasbian
 |     ├Makefile.ubuntu14
 |     └rtmouse.c
 ├supplement
 |  ├201507
```

```
|   |   ├Buzzer_circuit.jpg
|   |   ├Drive_circuit4.jpg
|   |   └ハードの簡単な説明.rst
|   ├201508
|   |   └RPiMouse-schematic.pdf
|   └201509
|       └MEMO.raspi-config.rst
└utils
    ├build_install.raspbian.bash
    └build_install.ubuntu14.bash
```

RaspberryPiMouse/lib/RaspberryPiのバージョン/カーネルバージョン/rtmouse.koがドライバ本体です。このファイルは「カーネルモジュール」といって、カーネルの一部として動作するプログラムの集まりです。また、SPI、I2Cをセンサで使用するのでSPI、I2C機能を有効にしてください。4章参照。

カーネルバージョン4.1.19以降のデバイスドライバでは、Raspberry Pi2とRaspberry Pi3共通で使用できます。Raspberry Pi3でデバイスドライバの組み込みはPi2+のデバイスドライバをご使用ください。

デバイスドライバの組み込みは “insmod”を使用します。”lsmod”というコマンドを打つと組み込み済みのカーネルモジュールがリストされるので、その中にrtmouseあることを確認してください。デバイスドライバが組み込まれたときにブザがなり、センサが光ります。

インストール
$sudo insmod rtmouse.ko⏎
メッセージが出なければ成功

Raspberry Pi のバージョン

または、
$sudo insmod /home/pi/RaspberryPiMouse/lib/Pi?B+/`uname -r`/rtmouse.ko⏎

カーネルのバージョン

インストールされているかの確認
$lsmod | grep rtmouse⏎
rtmouse 28672 0

インストールがうまくいったら、下記のように、/dev/のディレクトリの下にrt<なんとか>というファイルができているはずです。/dev/の下にあるファイルは「デバイスファイル」といわれるもので、機械と入出力をつかさどるファイルです。

```
$ls /dev/rt*⏎
/dev/rtbuzzer0    /dev/rmotor_raw_r0
/dev/rtled0       /dev/rtmotoren0
(略)
```

パーミッションを変更しておくことでルートにならずにデバイスドライバを扱うことができます。参考までに、ドライバをアンインストールする場合は、「rmmod」を使用します。

```
$sudo chmod 666 /dev/rt*⏎
$ls –l /dev/rt*⏎
crw-rw-rw- 1 root root 245, 0  4 月 27 14:38 /dev/rtbuzzer0
crw-rw-rw- 1 root root 246, 0  4 月 27 14:38 /dev/rtled0
crw-rw-rw- 1 root root 246, 1  4 月 27 14:38 /dev/rtled1
(以下略)
```

5.1 デバイスドライバのビルドの方法

Raspbian の OS が日々進化しているため、カーネルバージョンのアップデートを要求されることがあります。現状のままで問題ない方や、まだ Raspbian の OS がよくわからない方は、アップデートをしないことをお勧めします。アップデートして対応するカーネルバージョンがない場合、デバイスドライバをビルドする必要があります。デバイスドライバをビルドするには、カーネルヘッダというものが必要になります。新しいカーネルがリリースされて数ヶ月後にカーネルヘッダがビルドされたものがリリースされています。初めに、デバイスドライバをビルドするために必要なパッケージをインストールします。

```
sudo apt-get install bc bison flex libssl-dev make
```

カーネルヘッダをダウンロードします。

```
sudo apt-get install raspberrypi-kernel-headers
```

Makefile にあるパスを修正します。

```
vi ~/RaspberryPiMouse/src/drivers/Makefile.raspbian
```

make -C /usr/src/linux M=$(PWD) modules
を
make -C /usr/src/linux-headers-`uname -r` M=$(PWD) modules
に変更

RaspberryPiMouse/src/drivers にある Makefile は、ubuntu にリンクされているので、raspbian の方にリンクを付け直します。

```
cd ~/RaspberryPiMouse/src/drivers
rm Makefile
ln -s Makefile.raspbian Makefile
```

make を実行して、デバイスドライバをビルドします。

```
make
```

成功すると、/home/pi/RaspberryPiMouse/src/drivers に rtmouse.ko ができます。

6 デバイスドライバの使い方について

デバイスドライバでできることは、LED の点灯、ブザの発信、光反射型距離センサの値の取得、モータを回す、スイッチの on/off、パルスのカウントの確認ができます。

6.1 LED の操作

LED は 0～3 まであり、下記に示すコマンドは例として LED0 の点灯/消灯を示します。LED1 の場合は、rtled1 で点灯/消灯できます。最後の数字を 0～3 に変更することで LED の点灯個所が変わります。

```
点灯
        $ echo 1 > /dev/rtled0⏎
消灯
        $echo 0 > /dev/rtled0⏎
```

図 6-1 LED 操作

6.2 ブザの操作

echo で周波数(単位:Hz)を渡します。

```
440Hz を鳴らす
        $ echo 440 > /dev/rtbuzzer0⏎

消音する
        $ echo 0 > /dev/rtbuzzer0⏎
```

図 6-2 Buzzer 操作

参考までに音階の周波数を表 1 に示します。

小数点が使えません(0 と正の整数のみ)ので、切り上げ、切り下げはユーザの判断でお願いします。

表 1 音名(階名)と周波数一覧

オクターブ	音名(階名)											
	C(ド)	ド#	D(レ)	レ#	E(ミ)	F(ファ)	ファ#	G(ソ)	ソ#	A(ラ)	ラ#	B(シ)
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	111.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902
9	8372	8870	9397	9956	10548	11175	11840	12544	13290	14080	14917	15804

6.3 モータの操作

モータを回すには、ハードウェアスイッチ(MOTOR SW を ON)と、ソフトウェアスイッチ(/dev/rtmotoren0)を両方 ON にして初めてモータを回すことができます。プログラムがミスなくソフトウェアスイッチを ON/OFF できればハードウェアスイッチを付ける必要はないのですが、Raspberry Pi の暴走、ソフトウェアのバグなどによりソフトウェアスイッチで OFF できない場合、強制的に停止できるよう、ハードウェアスイッチを付けてあります。

```
ソフトウェアスイッチを ON
$ echo 1 > /dev/rtmotoren0⏎

左のモータのみを順方向に 400Hz で回す
$ echo 400 > /dev/rtmotor_raw_l0⏎

逆方向に 400Hz で回す
$ echo -400 > /dev/rtmotor_raw_l0⏎

右のモータのみを順方向に 250Hz で回す
$ echo 250 > /dev/rtmotor_raw_r0⏎

回転を止める
$echo 0 > /dev/rtmotor_raw_r0⏎
$echo 0 > /dev/rtmotor_raw_l0⏎
```

図 6-3 モータの操作(1/2)

左右のモータを与えた周波数で与えた時間回す
左 500Hz、右 10Hz、１秒間回す。時間の単位は ms
　　左　右　時間
$echo 500 10 1000 > /dev/rtmotor0

ソフトウェアスイッチを OFF
$ echo 0 > /dev/rtmotoren0⏎

図 6-3 モータの操作(2/2)

入力できる数字は整数のみです。
Raspberry Pi Mouse に使用されているモータはステッピングモータというもので、1 パルスで 0.9[deg]回り、400 パルスで一回転します。echo の後の数字は周波数を入力するようになっており、400 の場合は、1 秒間に 400 パルス入力され、丁度 1 秒間に一回転することになります。

6.4 センサの読み取り

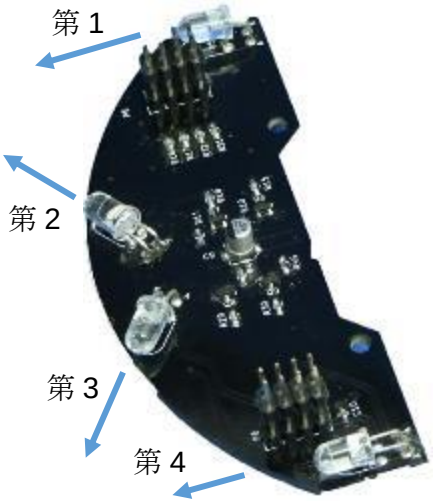


図 6-4 基板についている四つの可視光 LED

$ cat /dev/rtlightsensor0⏎
187 189 184 191
第 1、第 2、第 3、第 4 の順番でセンサの値が表示されます。
　値が表示されない場合は、SPIが有効になっていない可能性がありますので、4章参照してSPIを有効にしてください。

6.5 タクトスイッチの読み取り

タクトスイッチは 0～2 まであり、下記に示すコマンドは例として SW0 が押されたか押されていないかを示します。
タクトスイッチ 1 の場合は、rtswitch1 で最後の数字を 0～2 に変更することで各々のタクトス

イッチの状態を取得できます。

```
押されている状態
$ cat  /dev/rtswitch0⏎
0

押されていない状態
$cat  /dev/rtswitch0⏎
1
```

図 6-5 スイッチの操作

6.6 パルスカウンタの使い方

パルスカウンタは、I2C を使用しますので、I2C を有効にする必要があります。有効にする方法は、4 章を参照してください。パルスカウンタは、rtcounter_l0 と rtcounter_r0 の左右あります。rtmotor_raw_r0、rtmotor_raw_l0、rtmotor0 で生成されるパルスをカウントします。モータのソフトウェアスイッチの rtmotoren が 0 の状態でも、echo 400 400 1000 > /dev/rtmotor0 を実行すると、

```
$cat /dev/rtcounter_r0
412
```

とタイヤが回っていなくともパルスをカウントする仕様になっています。

```
カウントされているパルスを確認する場合
$ cat  /dev/rtcounter_r0⏎
100

カウンタをリセットする場合
$echo 1000 /dev/rtcounter_r0⏎
$cat /dev/rtcounter_r0
1000
```

図 6-6 パルスカウンタの操作

7 デバイスドライバの使用例

6 章でデバイスドライバの簡単な使い方を記述しましたが、この章では、Shell、C 言語、python、C++の 4 種類で具体的なプログラムを提示し、解説します。

7.1 Step1 LED を光らせよう

Step1 は表示用の LED を光らせてみます。LED を 0.5 秒ごとに点滅させるプログラム例を以下に示します。

7.1.1 Shell バージョン

```
#!/bin/bash
while true
do
        echo 1 | tee /dev/rtled?        #点灯
        sleep 0.5                       #0.5 秒待ち
        echo 0 | tee /dev/rtled?        #消灯
        sleep 0.5                       #0.5 秒待ち
done
```

図 7-1 step1 shell バージョン

動作確認と解説

上記のプログラムを step1.sh というファイルに保存し、実行の権限を追加(chmod +x step1.sh)し「$ ./step1.sh」で実行すると LED が点滅します。終了するには「Ctrl + C ⏎」で終了します。

無限に繰り返すコマンドとして while を使用しています。

while 文の書式

```
while 条件式
do
        処理
done
```

図 7-2 while の書式

条件が真の場合 do～done の間の処理を繰り返し実行します。”true”の代わりに「:」のヌルコマンドを指定しても無限ループになります。

デバイスドライバにデータを送るコマンドとして echo を使用しています。echo コマンドは、引数に指定された文字列や変数を表示します。

```
echo 1 > /dev/rtled0
```

とすれば、デバイスドライバ LED0 に”1”が送られ、LED が点灯します。”0”を送ると消灯します。「 > 」はリダイレクションと呼ばれるもので、左のデータを右のファイル(デバイスドライバ)に新規に書き込みます。
追加で書き込む場合は「>>」になります。ここの例題では「>」を使わずに tee コマンドと「?」(任意の 1 文字のワイルドカード、文字列の場合は「 * 」)を使用して LED0 から LED3 まで書き込むようにしています。
tee を使用するには、コマンドを連結するパイプ「 | 」を使用します。パイプを使用すると、コマンドは左から実行され、コマンドの実行結果を次のコマンドに渡されます。「echo 1」で発行された「1」が 「tee /dev/rtled?」に渡されます。「tee」はコマンドの結果をファイル(今回はデバイスドライバ)に出力しながら標準出力をするときに使用します。

sleep は指定した時間だけ停止します。記号なしの指定時間の単位は「秒」です。

7.1.2 C 言語　バージョン

```
#include "fcntl.h"

void main(void)
{
int led[4];
int i;


led[0] = open("/dev/rtled0",O_WRONLY);
led[1] = open("/dev/rtled1",O_WRONLY);
led[2] = open("/dev/rtled2",O_WRONLY);
led[3] = open("/dev/rtled3",O_WRONLY);

while(1)
{
        for(i=0;i<4;i++)
        {
                write(led[i],"1",1);
        }
        usleep(500*1000);
        for(i=0;i<4;i++)
        {
                write(led[i],"0",1);
        }
        usleep(500*1000);
}
        for(i=0;i<4;i++)
        {
                close(led[i]);
        }

}
```

図 7-3 step1 C 言語バージョン

動作確認と解説

上記のプログラムを step1.c というファイルに保存し、「$gcc step1.c –o step1」でコンパイルします。「-o」をしない場合は、「a.out」になります。コンパイルで作った実行ファイルを「$ ./step1」で実行すると LED が点滅します。終了するには「Ctrl + C ⏎」で終了します。

C 言語でファイルの読み書きに open、close 関数を使用します。ファイルのアクセスに限ら

ず、デバイスドライバにも使用できます。open、close 関数を使用するには ”fcntl.h” が必要になるため include します。
無限に繰り返すコマンドして while を使用しています。

while 文の書式

```
while (条件式)
{
        処理;
}
```

図 7-4 while の書式

条件が真の場合 {　} の間の処理を繰り返し実行します。似たようなコマンドで「do { } while(条件式); 」がありますが、「while(条件式) { } 」とは違い条件式が false でも一度は実行されるという仕様になります。つまり、「while(false){ }」では{ }内を一度も実行されないが、「do { } while(false);」では一度点滅が実行されます。

```
while(0){
    write (led0,"1",1);
    usleep(500 * 1000);
    write (led0,"0",1);
    usleep(500 * 1000);
}
```

図 7-5 LED0 が一度も点滅しない

```
do{
    write (led0,"1",1);
    usleep(500 * 1000);
    write (led0,"0",1);
    usleep(500 * 1000);
} while(0)
```

図 7-6 LED0 が一度だけ点滅する

デバイスドライバに出力するコマンドして write コマンドを使用しています。
write の書式は

```
int write(int fd, void *buf, unsigned int n);
```

図 7-7 wrtie の書式

int fd:データを書き込むファイルを指定します。この例では、デバイスドライバになります。
void *buf:書き込むデータのアドレスを指定します。この例では、直接データを書き込んでいます。
unsigned n:データの大きさを指定します。この例では 1 文字を出力するので 1 としています。

fwrite でもデバイスドライバに書き込むことが可能ですが、fwrite の場合、内部の buffer を経由するため、buffer に溜まる または close されるまで出力されません。

C 言語には sleep 関数がありますが、小数点が使えず、マイクロ秒[μs]の usleep を使って 0.5 秒の sleep を実現しています。単位は、10^{-3} が ms、10^{-6} がμs でμs から ms にするには 1000

倍する必要があり、0.5 秒　= 500ms なので、500　* 1000 で 0.5 秒となります。

7.1.3　Python バージョン

```
#!/usr/bin/python
import time
import sys

files = [ "/dev/rtled0","/dev/rtled1","/dev/rtled2","/dev/rtled3"]

while 1:
	for filename in files:
		with open(filename,'w') as f:
			f.write("1")


	time.sleep(0.5)

	for filename in files:
		with open(filename,'w') as f:
			f.write("0")

	time.sleep(0.5)
```

図 7-8 step1 python バージョン

動作確認と解説

上記のプログラムを step1.py というファイルに保存し、「$python step1.py 」で実行すると LED が点滅します。終了するには「Ctrl + C ⏎」で終了します。

デバイスドライバに書き込むコマンドとして write を使用しています。write 単体で使用するとバッファに入るため、ある程度データの書き込みをしないとデバイスドライバに書いてくれません。with を使うことで自動的に close をしてくれるので、バッファに溜まらずデータがすぐに反映されます。sleep 関数は time に含まれているため time を import しています。

C 言語では while 内で処理するものとして「{ }」を使用していましたが、Python は「{ }」を使用せず、インデントで制御します。

7.1.4 C++バージョン

```
#include <cstdio>
#include <unistd.h>

using namespace std;

void set0( bool data )
{
FILE *led;
	led = fopen("/dev/rtled0", "w" );

	if ( data ){
		fprintf( led, "1" );
	} else {
		fprintf( led, "0" );
}

	fclose( led );
}

void set1( bool data )
{
	FILE *led;
	led = fopen("/dev/rtled1", "w" );

	if ( data ){
		fprintf( led, "1" );
	} else {
		fprintf( led, "0" );
	}

	fclose( led );
}

void set2( bool data )
{
	FILE *led;
	led = fopen("/dev/rtled2", "w" );

	if ( data ){
		fprintf( led, "1" );
	} else {
		fprintf( led, "0" );
	}

	fclose( led );
}
```

図 7-9 step1 C++バージョン(1/2)

```
void set3( bool data )
{
	FILE *led;
	led = fopen("/dev/rtled3", "w" );

	if ( data ){
		fprintf( led, "1" );
	} else {
		fprintf( led, "0" );
	}

	fclose( led );
}

int main()
{

	while( 1 )
	{
		set0( true );
		set1( true );
		set2( true );
		set3( true );

		usleep( 500 * 1000 );

		set0( false );
		set1( false );
		set2( false );
		set3( false );

		usleep( 500 * 1000 );

	}

	return 0;
}
```

図 7-9 step1 C++バージョン(2/2)

動作確認と解説

上記のプログラムを step1.cpp というファイルに保存し、「$g++ step1.cpp –o step1」でコンパイルします。「-o」をしない場合は、「a.out」になります。コンパイルで作った実行ファイルを「$ ./step1」で実行すると LED が点滅します。終了するには「Ctrl + C ⏎」で終了します。

C++の言語の仕様として C 言語と同様の書き方をすることができます。

C 言語バージョンで紹介されている、ファイル入出力での操作となります。C++では、open,write,close 関数が使えません。

setvbuf 関数を使用することで、バッファをなくすことはできますが、状況次第でエラーが起きる可能性が高くなるため、毎回ファイルを開いて書き込んで閉じています。また、プログラムを見やすくするために関数にまとめています。

7.2 Step2 ブザを鳴らそう

デバイスドライバを使用すると周波数をデバイスドライバに送るだけで音が鳴ります。キーボードをたたいてピアノのように音階を出すプログラム例を以下に示します。
キーボードと音階の配置の仕様は以下のようになっています。

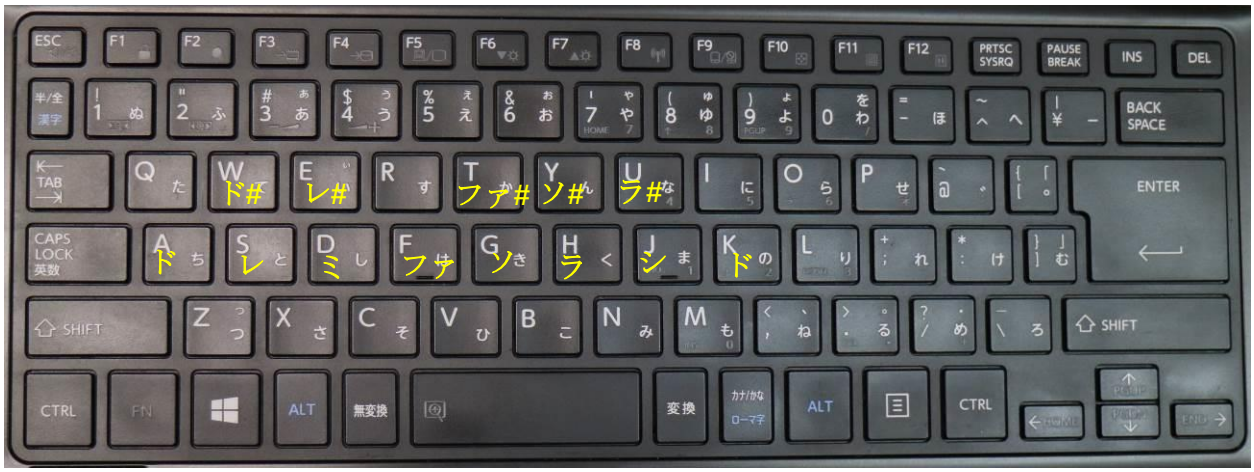


図 7-10 step2 音階

7.2.1 Shell バージョン

step2.sh

```
#!/bin/bash
while read -N 1 b ; do
        awk -v "t=$b" '$3==t{print $2}' SCALE > /dev/rtbuzzer0
done
```

図 7-11 step2 shell バージョン(1/2)

SCALE ファイル

```
off 0 0
ド 261 a
ド# 277 w
レ 293 s
レ# 311 e
ミ 329 d
ファ 349 f
ファ# 370 t
ソ 392 g
ソ# 415 y
ラ 440 h
ラ# 466 u
シ 493 j
ド 523 k
```

図 7-11 step2 shell バージョン(2/2)

動作確認と解説

上記のプログラムを 1 つ目を step2.sh、2 つ目を SCALE というファイル名で保存し、実行の権限を追加し「$ ./step2.sh」で実行すると、キーボードを叩くと音が鳴ります。

Shell には連想配列がないため、awk と組み合わせて連想配列を実現しています。while の後の「read –N 1 b」はキーボードから 1 文字読んで変数 b に代入しています。「-N 数字」のオプションで変数に入れる文字数を指定できます。「-N 数字」のオプションがない場合は、リターンを押すまで文字数が変数に入力されます。

7.2.2 C 言語バージョン

```
#include "fcntl.h"

int _Getch(void)
{
	int ch;
	system("stty -echo -icanon min 1 time 0");
	ch = getchar();
	system("stty echo icanon");
	return ch;
}

void main(void){
	int buzzer= open("/dev/rtbuzzer0" , O_WRONLY  );
	int c=1;

	while(c){
		switch(_Getch()){
		case '0'://off
			write (buzzer,"0",2);
			break;
		case 'a'://do
			write(buzzer,"261",4);
			break;
		case 'w'://do#
			write(buzzer,"277",4);
			break;
		case 's'://re
			write(buzzer,"293",4);
			break;
		case 'e'://re#
			write(buzzer,"311",4);
			break;
		case 'd'://mi
			write(buzzer,"329",4);
			break;
		case 'f'://fa
			write(buzzer,"349",4);
			break;
```

図 7-12 step2 C 言語バージョン(1/2)

```
                case 't'://fa#
                        write(buzzer,"370",4);
                        break;
                case 'g'://so
                        write(buzzer,"392",4);
                        break;
                case 'y'://so#
                        write(buzzer,"415",4);
                        break;
                case 'h'://ra
                        write(buzzer,"440",4);
                        break;
                case 'u'://ra#
                        write(buzzer,"446",4);
                         break;
                case 'j'://shi
                        write(buzzer,"493",4);
                         break;
                case 'k'://DO
                         write(buzzer,"523",4);
                        break;
                case ‘c’://プログラムを終了
                        write(buzzer,”0”,2);
                        c=0;
                        break;
                }
        }

        close(buzzer);

}
```

図 7-12 step2 C 言語バージョン(2/2)

動作確認と解説

上記のプログラムを step2.c というファイル名で保存し、「$gcc step2.c –o step2」でコンパイルします。「$ step2」で実行し、キーボードを叩くと音がなります。

C 言語の getchar コマンドはリターンキーを叩いたときに入力データが変数に更新されるため、音階に指定したキーボードを叩くだけでは音がなりません。そこで、system 関数を使って Linux 上でエコーバックをしない(-echo)、特殊文字を無効にする(-icanon)、-canon が設定されているとき、最小文字数が読み込まれなかった場合に時間切れにするタイム(time N)、-canon が設定されているとき、最低 N 文字(min N)を設定しています。これにより、一文字入力された

らすぐに変数に反映されます。

7.2.3 Python バージョン

```
#!/usr/bin/python
import sys

class _Getch:
    def __init__(self):
        import tty, sys

    def __call__(self):
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch

dict= { "0":"0", "a":"261", "w":"277", "s":"293","e":"311","d":"329","f":"349","t":"370",
"g":"392","y":"415","h":"440","u":"446","j":"493","k":"523"}

while 1:
        getch = _Getch()
        d = getch()
        if d == "c" :
                break
        if d in dict :
                with open('/dev/rtbuzzer0','w') as f:
                        f.write(dict[d])
```

図 7-13 step2 python バージョン

動作確認と解説

上記のプログラムを step2.py というファイル名で保存し、「$ python step2.py 」で実行し、キーボードを叩くと音がなります。終了するには「Ctrl + C ⏎」または、”c”を入力すると終了します。

Python には、連想配列(ディクショナリ)があり、それを使用することですっきりしたプログラムになります。Python にはキーボードの入力を取り込む「raw_input()」コマンドがありますが、「raw_input()」コマンドはキーボードをたたいた後リターンキー⏎を押さないとデータの取り込みができない仕様となっているため、_Getch 関数を作り、キーボードをたたいた後すぐにデータの取り込みができるようにしています。

7.2.4 c++バージョン

```
#include <cstdio>
#include <unistd.h>

using namespace std;

void off()
{
        FILE *bz;
        bz = fopen( "/dev/rtbuzzer0", "w" );
        fprintf( bz, "0" );
        fclose( bz );
}

void on( const char *scale, int wait_time )
{
        FILE *bz;
        bz = fopen( "/dev/rtbuzzer0", "w" );
        fprintf( bz, scale );
        fclose( bz );
        usleep( wait_time * 1000 );
        off();
}

int main()
{
        on( "440", 500 );
        return 0;
}
```

図 7-14 step2 C++バージョン

動作確認と解説

上記のプログラムを step2.cpp というファイル名で保存し、「$ g++ step2.cpp –o step2」でコンパイルします。「$ step2」で実行すると、ブザから 440Hz の音が 500ms の間一度なります。C 言語の場合と同じ動作をさせたい場合は、main 内で下記のようにすることで可能です。

```
while( 1 ){
    switch( getcahr() ){
      case ‘a’:
        on( “440”, 500);
        break;
    ケースをつくる
  }
}
```

図 7-15 step2 C++バージョン (キー入力あり)

標準入出力の関数を使用するため、c 言語よりも動作が遅くなることが考えられます。

7.3 Step3 スイッチを使おう

スイッチ(SW0)を押すと LED3 を点灯/消灯を繰り返し、スイッチ(SW1)を押すと LED1、LED2 の点灯/消灯を繰り返し、スイッチ(SW2)を押すと LED0 を点灯/消灯を繰り返すプログラムを以下に示します。

7.3.1 Shell バージョン

```
#!/bin/bash

state0=0
state1=0
state2=0

while true ; do
        if grep -q 0 /dev/rtswitch0 ; then
                sleep 0.1
                while grep -q 0 /dev/rtswitch0 ; do
                        sleep 0.1
                done
                state0=`expr $state0 + 1`
                state0=`expr $state0 % 2`
                echo $state0 > /dev/rtled3
        fi
        if grep -q 0 /dev/rtswitch1 ; then
                sleep 0.1
                while grep -q 0 /dev/rtswitch1 ; do
                        sleep 0.1
                done
                state1=`expr $state1 + 1`
                state1=`expr $state1 %2`
                echo $state1 > /dev/rtled2
                echo $state1 > /dev/rtled1
        fi
```

図 7-16step3 shell バージョン(1/2)

```
        if grep -q 0 /dev/rtswitch2 ; then
            sleep 0.1
            while grep -q 0 /dev/rtswitch2 ; do
                sleep 0.1
            done
            state2=`expr $state2 + 1`
            state2=`expr $state2 % 2`
            echo $state2 > /dev/rtled0
        fi
    done
```

図 7-16 step3 shell バージョン(2/2)

動作確認と解説

上記のプログラムを step3.sh というファイル名で保存し、実行の権限を追加し「$ ./step3.sh」で実行すると、タクトスイッチ(SW0～SW2)を押すと LED が光ります。タクトスイッチが押されると”0”が入力されます。押されていない状態では”1”が入力されます。if grep –q 0 /dev/rtswitch??で 0 が入力されたあと sleep 0.1 で少し待っています。これは、機械的スイッチを押したとき/離したときに内部の金属が震えている状態になります。この状態をチャタリングといいます。チャタリング時にスイッチの状態を調べると”0”か”1”のいずれの値も取れてしまうため、押したのか離したのかが分かりません。しかし、チャタリングは、数μs から数 ms 待つことによって固定されますので、sleep 0.1 でチャタリングが終わった後スイッチの状態を確認しに行っています。

7.3.2 C 言語 バージョン

```
#include "fcntl.h"

char get_SW0(void){
    char buf[2];
    int SW0;
    SW0 = open("/dev/rtswitch0",O_RDONLY);
    read(SW0,buf,2);
    close(SW0);
    return buf[0];
}

char get_SW1(void){
    char buf[2];
```

図 7-17 step3 C 言語バージョン(1/3)

```
        int SW1;
        SW1 = open("/dev/rtswitch1",O_RDONLY);
        read(SW1,buf,2);

        close(SW1);
        return buf[0];
}

char get_SW2(void){
        char buf[2];
        int SW2;
        SW2 = open("/dev/rtswitch2",O_RDONLY);
        read(SW2,buf,2);
        close(SW2);
        return buf[0];
}

void main(void){
        int state0,state1,state2;
        int LED0,LED1,LED2,LED3;

        LED0 = open("/dev/rtled0",O_WRONLY);
        LED1 = open("/dev/rtled1",O_WRONLY);
        LED2 = open("/dev/rtled2",O_WRONLY);
        LED3 = open("/dev/rtled3",O_WRONLY);

        state0 = state1 = state2 = 0;

        while(1){
                if (get_SW0() =='0'){
                        usleep(10000);
                        while (get_SW0() == '0');
                        usleep(10000);
                        state0=(state0+1)&0x01;
                        if(state0==0){
                                write(LED3,"0",1);
                        }else{
                                write(LED3,"1",1);
                        }
                }
```

図 7-17 step3 C 言語バージョン(2/3)

```
                if (get_SW1() =='0'){
                        usleep(10000);
                        while (get_SW1() == '0');
                        usleep(10000);
                        state1=(state1+1)&0x01;
                        if(state1==0){
                                write(LED2,"0",1);
                                write(LED1,"0",1);
                        }else{
                                write(LED2,"1",1);
                                write(LED1,"1",1);
                       }
                }
                if (get_SW2() =='0'){
                        usleep(10000);
                        while (get_SW2() == '0');
                        usleep(10000);
                        state2=(state2+1)&0x01;
                        if(state2==0){
                                write(LED0,"0",1);
                        }else{
                                write(LED0,"1",1);
                        }
                }
        }
    }
```

図 7-17 step3 C 言語バージョン(3/3)

動作確認と解説

上記のプログラムを step3.c というファイル名で保存し、「$ gcc step3.c –o step3」でコンパイルします。「$ step3」で実行し、スイッチを押すと LED が光ります。
/dev/rtswitch?からは、2 バイトの情報が送られてきます。1 つ目はスイッチの状態の”0”または”1”、2 つ目は改行コードです。read で 2 バイト受信し、1 つのデータを使用しています。main 関数内で、スイッチの入力を記述すると見にくくなるため関数化しています。

7.3.3 Python バージョン

```
#!/usr/bin/python
import time

state0=state1=state2=0

while 1 :
	with open("/dev/rtswitch0","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch0","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state0 = (state0 + 1 ) & 1
			with open("/dev/rtled3","w") as f:
				f.write(str(state0))
	with open("/dev/rtswitch1","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch1","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state1 = (state1 + 1 ) & 1
			with open("/dev/rtled2","w") as f:
				f.write(str(state1))
			with open("/dev/rtled1","w") as f:
				f.write(str(state1))
	with open("/dev/rtswitch2","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch2","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state2 = (state2 + 1 ) & 1
			with open("/dev/rtled0","w") as f:
				f.write(str(state2))
```

図 7-18 step3 python バージョン

動作確認と解説

上記のプログラムを step3.py というファイル名で保存し、「$ python step3.py」で実行し、スイッチを押すと LED が光ります。Python は数値を文字列に変換する”str”の組み込み関数があり、それを使用して state の値をそのまま LED に出力しています。

7.3.4 C++バージョン

```
#include <cstdio>
#include <unistd.h>
#include <iostream>

using namespace std;

bool get0()
{
  bool sw_data = false;
  FILE *sw;
  sw = fopen("/dev/rtswitch0", "r");
  char data = fgetc(sw);
  if ( data == '0' ){
    sw_data = true;
  } else {
    sw_data = false;
  }
  fclose(sw);
  return sw_data;
}

bool get1()
{
  bool sw_data = false;
  FILE *sw;
  sw = fopen("/dev/rtswitch1", "r");
  char data = fgetc(sw);
  if ( data == '0' ){
    sw_data = true;
  } else {
    sw_data = false;
  }
  fclose(sw);
  return sw_data;
}
```

```
bool get2()
{
  bool sw_data = false;
  FILE *sw;
  sw = fopen("/dev/rtswitch2", "r");
  char data = fgetc(sw);
  if ( data == '0' ){
    sw_data = true;
  } else {
    sw_data = false;
  }
  fclose(sw);
  return sw_data;
}

int main()
{
while( 1 )
  {
    cout << get0() << "," << get1() << "," << get2() << endl;
  }

  return 0;
}
```

図 7-19 step3 c++バージョン

動作確認と解説

上記のプログラムを step3.cpp というファイル名で保存し、「$g++ step3.cpp –o step3」でコンパイルします。「$ Step3」で実行し、ターミナルを確認すると[0,0,0]と表示がされ続けます。左から順にスイッチ 0,1,2 の状態を示しておりスイッチが押されると 1 に数字が変わります。終了するには「Ctrl + C ⏎」で終了します。

8 備考

8.1 参考文献

日経 Linux 2015 年 6 月号～10 月号「 Raspberry Pi で始めるかんたんロボット製作」
日経 BP ラズパイマガジン 2016 年春号

8.2 著作権

9 問い合わせ

本製品に関するお問い合わせは、下記までお願いします。

株式会社 アールティ
〒101-0021 東京都千代田区外神田 3-2-13 山口ビル 3F
Email:support@rt-net.jp
URL: http://www.rt-net.jp

10改版履歴

版数	改版内容	改版日
1.0	・リリース	2015 年 8 月 4 日
1.1	・図 3-1 のパスを修正 ・図 3-2 にアクセスの方法を追加 ・表 1 音階を追加	2015 年 9 月 15 日
2.0	・名称の図を RaspberryPiMouseV2 に変更 ・RasPiMouse の組み立て手順を追加 ・Raspbian の OS のインストール手順を追加 ・OS のインストールのところで初期ユーザとパスワードを記載 ・デバイスドライバのインストールのところでデバイスドライバが RasberryPi2 と RaspberryPi3 共通で使えることを追加 ・モータの操作で追加された機能 “echo 左 Hz 右 Hz 時間 ms > /dev/rtmotor0”のサンプルを記載 ・デバイスドライバの使用例を追加	2016 年 6 月 15 日
2.1	・c++のサンプルプログラムを追加 ・誤字脱字の修正	2019/3/25
3.0	・誤字の修正 ・パルスカウンタの使い方の追加 ・充電の方法を追加 ・DC ケーブルの使い方を追加 ・デバイスドライバのビルドの方法を追加 ・本体と RaspberryPi の接続方法を追加	2020/3/1