Jetson Nano Mouse

取り扱い説明書

1.0版
2020年09月14日
株式会社アールティ

Contents:

1.概要

本マニュアルはJetson Nano Mouseの組み立てから、ソフトウェア環境の構築、サンプルコードの扱い方を解説しています。

本マニュアルではキーボードでキーを同時押しする操作があります。例えば、Ctrl, Alt, Tの3つのキーを同時押しする場合、「Ctrl + Alt + T」のように表記します。

本マニュアルではCLIでの操作を行います。実行するコマンドは枠で囲んで説明しています。

```
$ uname -a
```

上記のように書かれていた場合、端末にuname -aと入力し、Enterキーを押します。$や$の次のスペースは入力しません。

```
$ uname -a
Linux ubuntu 4.4.0-1080-raspi2 #88-Ubuntu SMP Mon Dec 11 14:23:15 UTC
2017 armv7l armv7l armv7l GNU/Linux
```

上記のように実行結果とともに説明する場合もあります。この場合も端末にuname -aと入力し、Enterキーを押します。$や$の次のスペースは入力しません。続くLinux...の行はコマンドの実行結果です。こちらも入力しません。本マニュアルに記載されている実行結果とお手元の環境での実行結果が同様の内容になっていることを確認してください。なお、ソフトウェアのアップデートにより表示内容が変更される場合があります。

1.1 内容物

- Jetson Nano Mouse 本体 1台
- 接続コネクタ　1個
- M2.6-5mmのネジ　4個
- フラットケーブル　2個
- Li-Poバッテリ　1個

※NVIDIA Jetson Nano 開発者キット B01、microSDカード、充電器は別売です。

1.2 オプション品（別売）

- ロボット用充電器セット12V5A(RT-ACDC-VH-12V5A)

2.使用環境

microSDカードにJetson Nano Developer Kit SD Card Imageをダウンロードして書き込むためのPCが必要です。そのためStorageの空きが20GB以上必要です。
Imageをダウンロードするため、ネットワークに繋がるPCが必要です。また、microSDカードにイメージを書き込むため、microSDカードアダプタを用意してください。
PCからsshでJenson Nanoにログインするには、Jenson Nanoに有線LANで接続する必要があります。Wi-Fi接続するには、別売のWi-Fiのドングルをつける必要があります。

OS：Windows 10 (32/64bit)、macOS、Linux
CPU：800MHz以上の32bit(x86) or 64bit(x64)のプロセッサ
装置　　：microSDカードリーダ
Memory　　：4GB以上
Storage　　：128GB以上
ネットワーク環境：有線LAN、Wi-Fi

3.動作準備

3.1 準備するもの

1. Jetson Nano Mouse本体
2. Jetson Nano Mouse のコネクタ基板(付属)
3. M2.6-5mmのネジ (4本、付属)
4. Li-Poバッテリ(付属)または、ACアダプタ(別売)
5. NVIDIA Jetson Nano 開発者キット B01(別売)
6. microSDカード(32GB以上推奨 別売)
7. 1番のプラスドライバ
8. USBマウス&USBキーボード
9. HDMIモニタ

3.2 OSの環境構築

microSDカードにJetson Nano Developer Kit SD Card Imageを書き込みます。以下のサイトから最新版のJetson Nano Developer Kit SD Card Imageをダウンロードすることができます。

https://developer.nvidia.com/embedded/downloads

2020年9月9日現在最新版はR32.4.3(JetPack 4.4)ですが、9月9日時点で公開しているデバイスドライバは、R32.3.1(JetPack 4.3)に対応しています。
Jetpack4.3は、以下のサイトからダウンロードします。

https://developer.nvidia.com/jetpack-43-archive

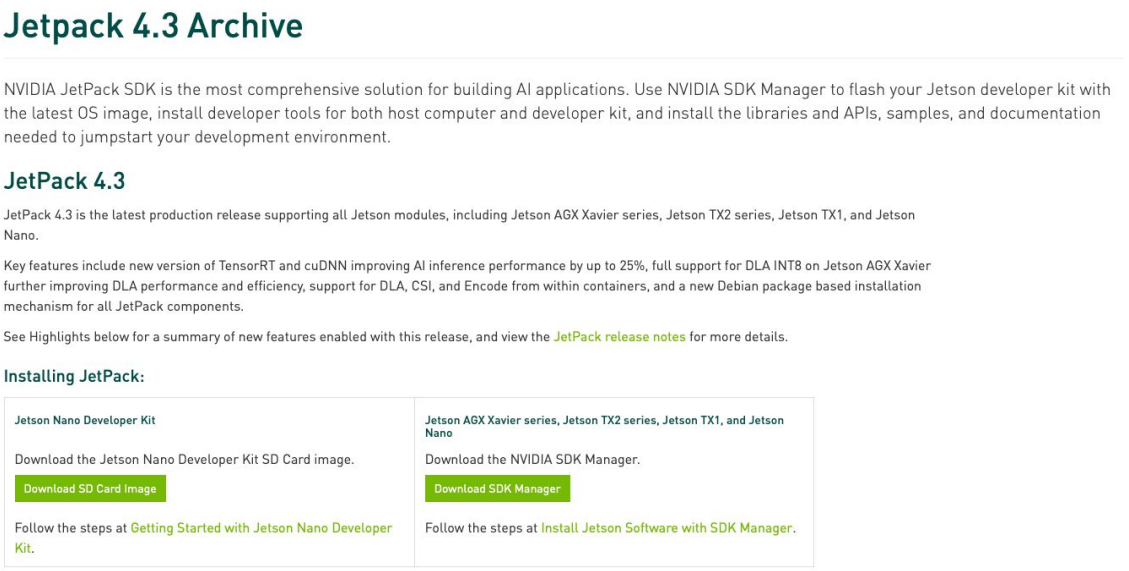

図3.1 NVIDIA JetPackダウンロードサイト

Jetson Nano Developer Kit(左側)のDownload SD Card Imageをクリックしてダウンロードします。約5GBあるので、ダウンロードが終わるまでしばらく待ちます。

イメージデータのダウンロードが終わったら、microSDカードに書き込みます。エクスプローラーやFinderなどのOS標準のツールでコピーするとイメージの書き込みではなく、ファイルに書き込みになってしまうため、OSが起動しません。そこで、Etcherなどのソフトを使ってダウンロードしたイメージファイルをmicroSDカードに書き込む必要があります。

balenaEtcherは、以下のサイトからダウンロードすることができます。

https://www.balena.io/etcher/

balenaEtcherは、Windows、macOS、Linuxに対応しています。

macOSで書き込みができない場合は、ターミナルからコマンドでsudoを付けてターミナルからEtcherを起動すると書き込むことができます。

Jetson Nanoにイメージを書き込んだmicroSDカードを挿入します。ロック機能が付いていますので、カチッと音がするまで押すとmicroSDカードが出ていない状態になります。取り出す際には、microSDカードを再度押して下さい。

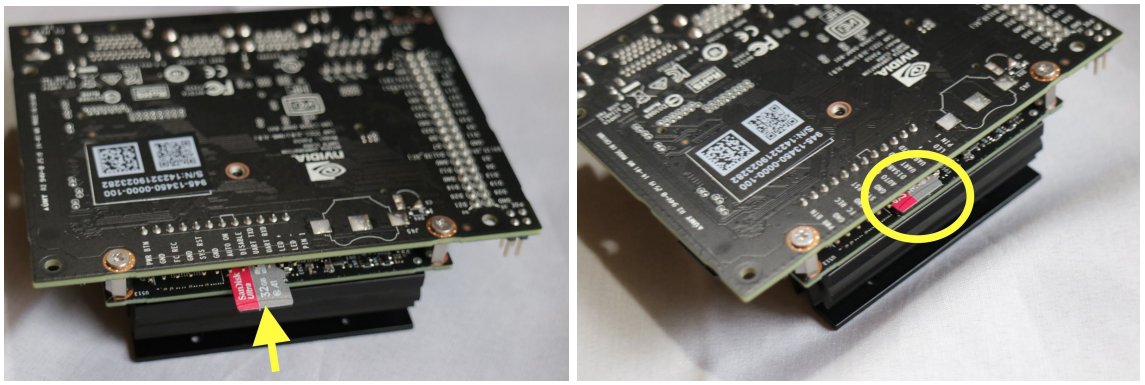

図3.2 microSDカードの挿入

4.ハードウェアについて

ここでは、Jetson Nano Mouseの各部の名称の確認と組み立てを行います。

4.1 パーツ一覧

NVIDIA Jetson Nano 開発者キット B01と本体の接続以外、組み立てられた状態で出荷されますので、必要なパーツは以下になります。

- 本体
- Jetson Nanoとつなぐ接続コネクタ
- NVIDIA Jetson Nano 開発者キット B01(別売)
- microSDカード(別売 32GB以上)

4.2 名称について

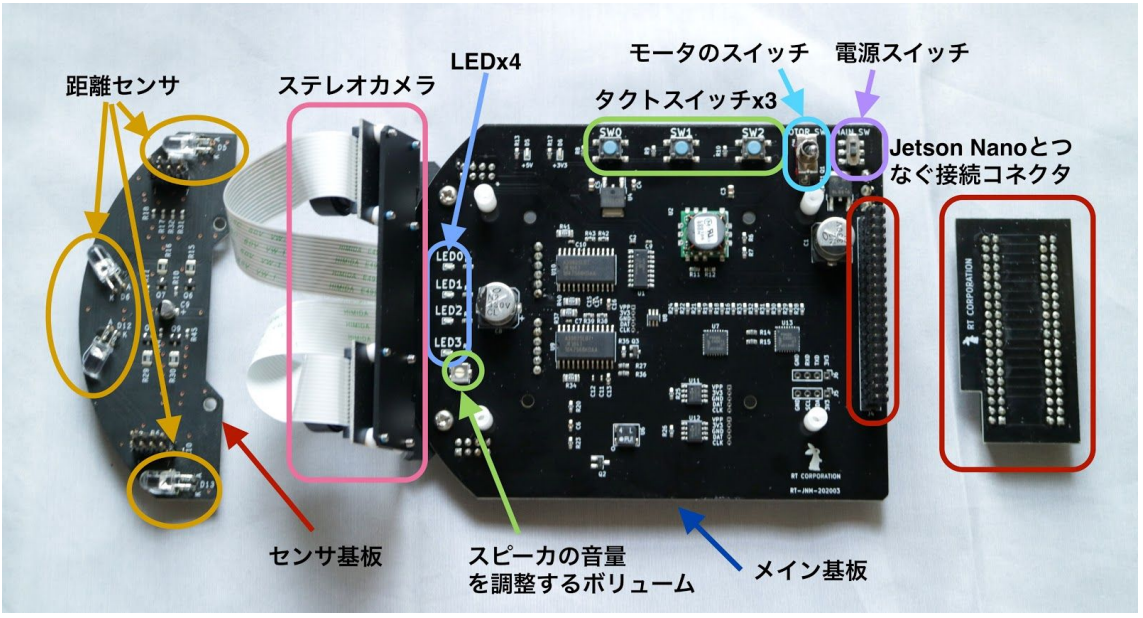

図4.1 名称

Jetson Nanoについているジャンパーピンは Jetson Nanoボードに直接ACアダプタから電源を供給するときに使用します。Jetson Nano Mouseは、メイン基板経由でJetson Nanoに電源を供給するため、ジャンパーピンを取り外してください。

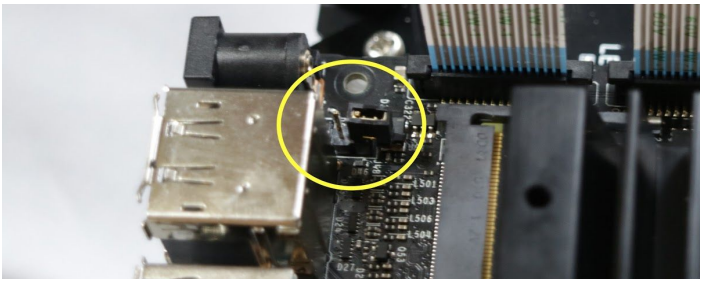

図4.2 ジャンパーピン取り外し

ステレオカメラのフラットケーブルをJetson Nanoに接続します。フラットケーブルの青色がカメラ側になるように接続します。コネクタを上に持ち上げるとフラットケーブルを入れることができます。力一杯上にあげてしまうと取れてしまうので注意してください。

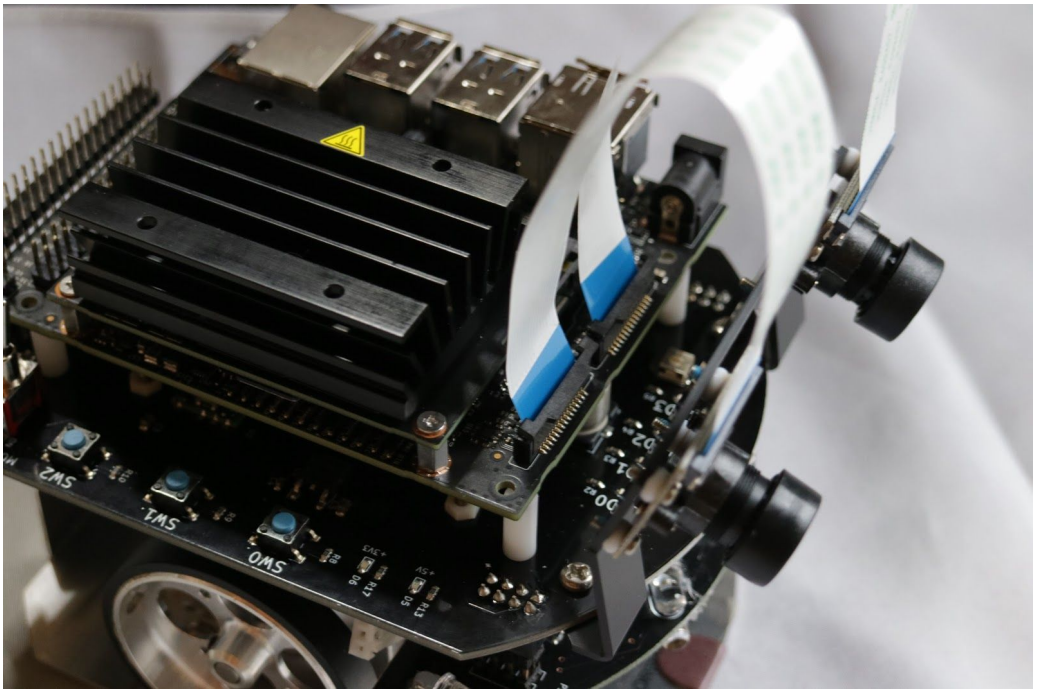

図4.3 フラットケーブルをJetson Nanoに挿す

フラットケーブルを固定するには、先ほど持ち上げたコネクタを押すとフラットケーブルが固定されます。

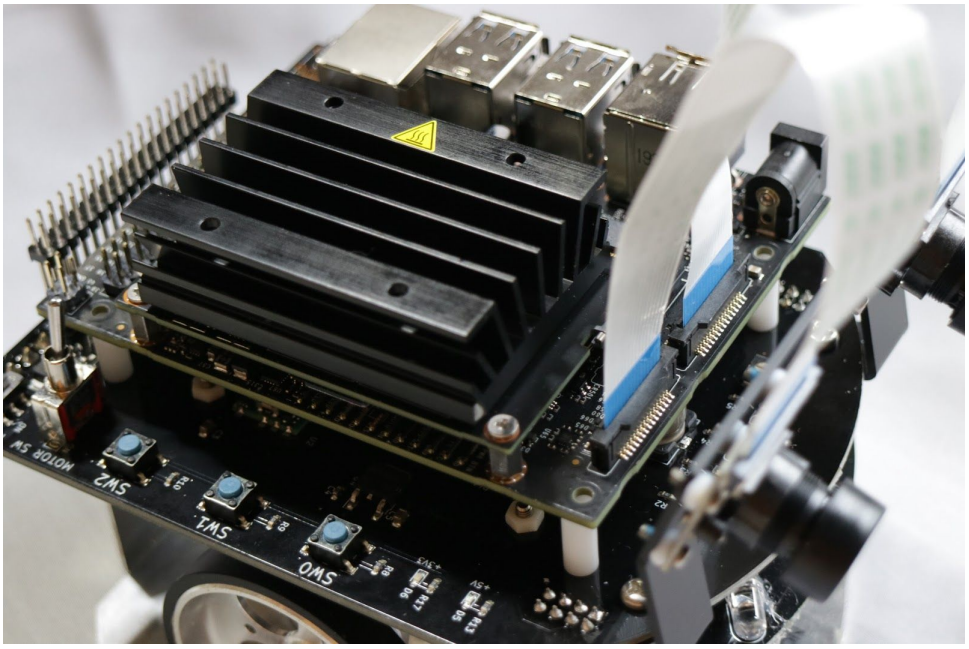

図4.4 Jetson Nanoのコネクタを押し込む

後ろから見るとコネクタとフラットケーブルに隙間がない状態になります。また、フラットケーブルは基板に対して垂直になるように入れてください。

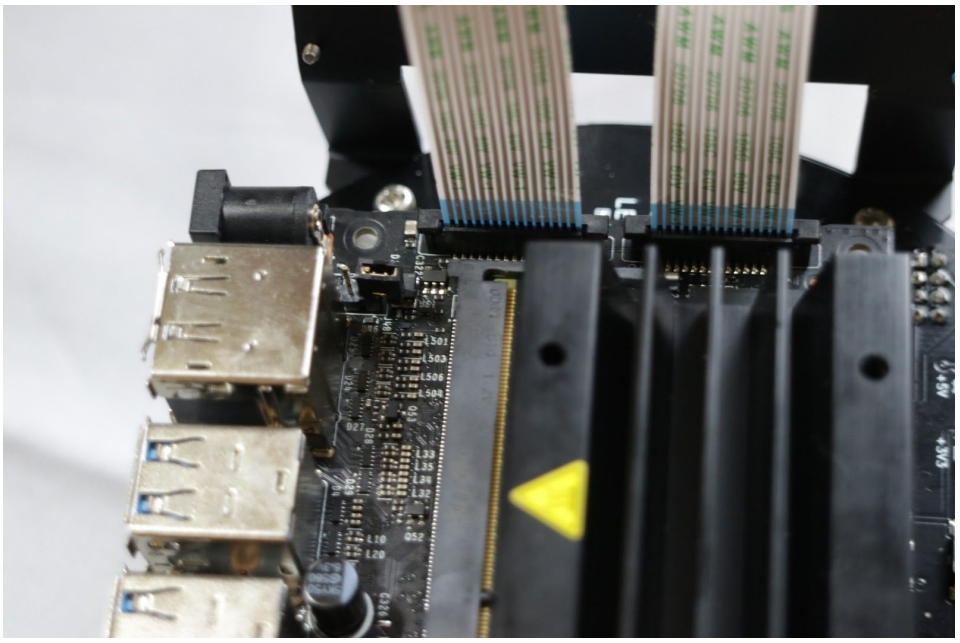

図4.5 フラットケーブルは垂直に

付属していたネジでJetson Nanoを固定します。１つずつネジで止めていくと最後のネジが入らないことがあるため、ネジが取れない程度に回した状態(2、3回まわした程度)で4箇所を軽く固定し、その後少しずつ満遍なく回して止めます。

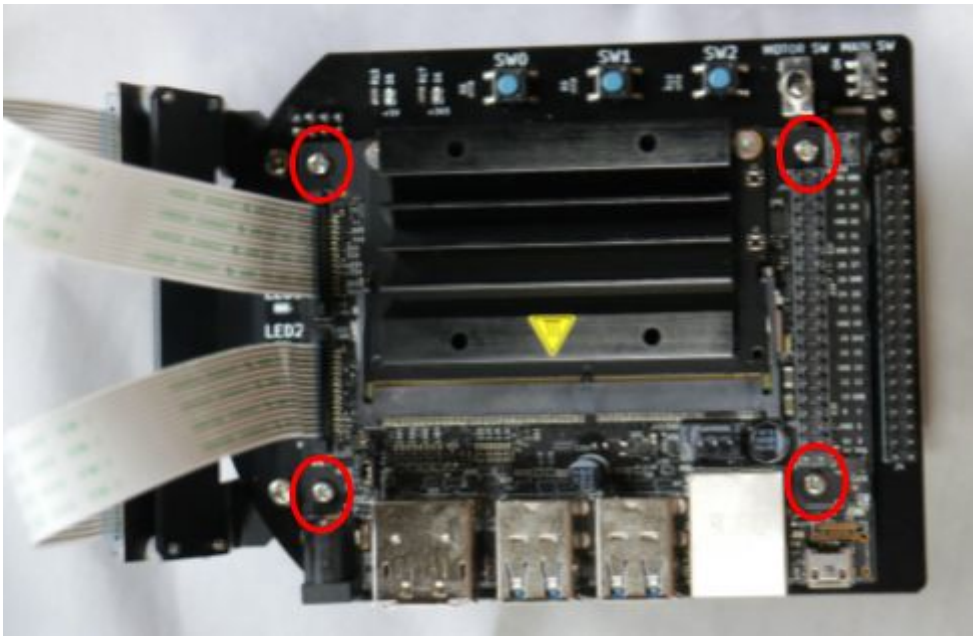

図4.6　Jetson Nano 固定のためのネジ締め

最後にコネクタをずらさないようにつけます。横から見てピンヘッダが出ていないことを確認してください。

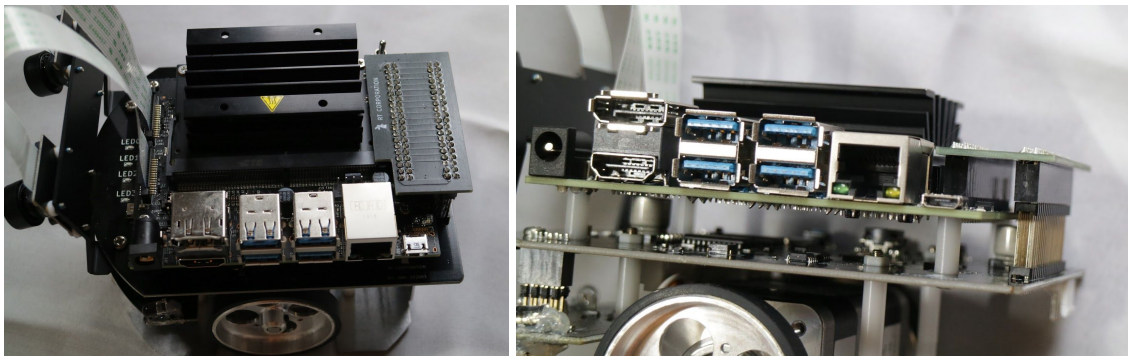

図4.7 接続コネクタ取り付け＆チェック

4.3 Li-Poバッテリの接続

Li-Poバッテリは、Jetson Nano Mouseの後方に図のように横の隙間からスライドさせて搭載し、コネクタを接続します。また、バッテリの充電用端子をテープなどで覆うなどして端子が金属部に触らないようにしておきます。Li-Poバッテリを接続する際は、Li-Poバッテリの容量が十分に残っていることを確認し、別紙の「電池に関するご注意」をよく読んだ上でご使用ください。

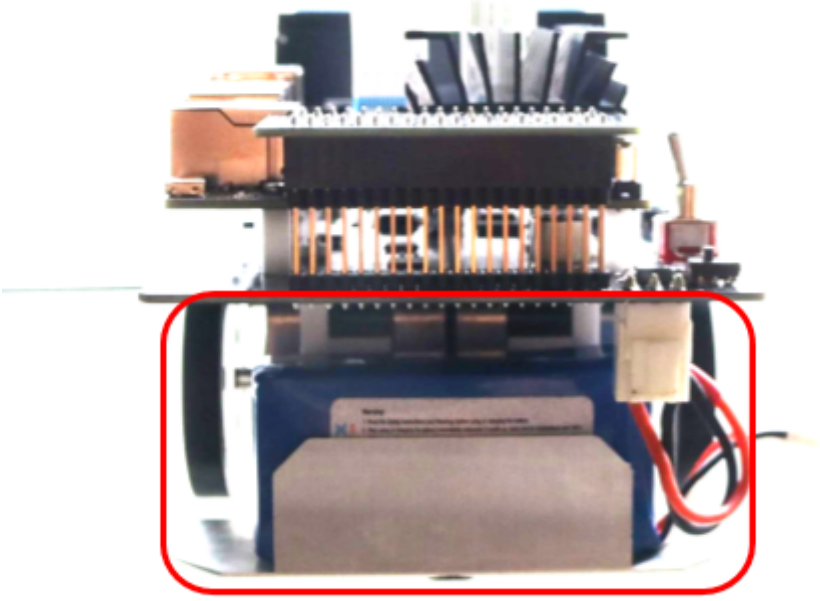

図4.8 Li-Poバッテリ接続

4.4 ACアダプタの接続

オプションのロボット用充電器セット12V5A(RT-ACDC-VH-12V5A)を購入された方は、Li-PoバッテリのかわりにACアダプタを使用することができます。ACアダプタを使用する場合は、最初に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。

図4.9 変換ケーブル

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。**Jetson Nano Mouseに接続した後に、ACアダプタをコンセントに接続**してください。

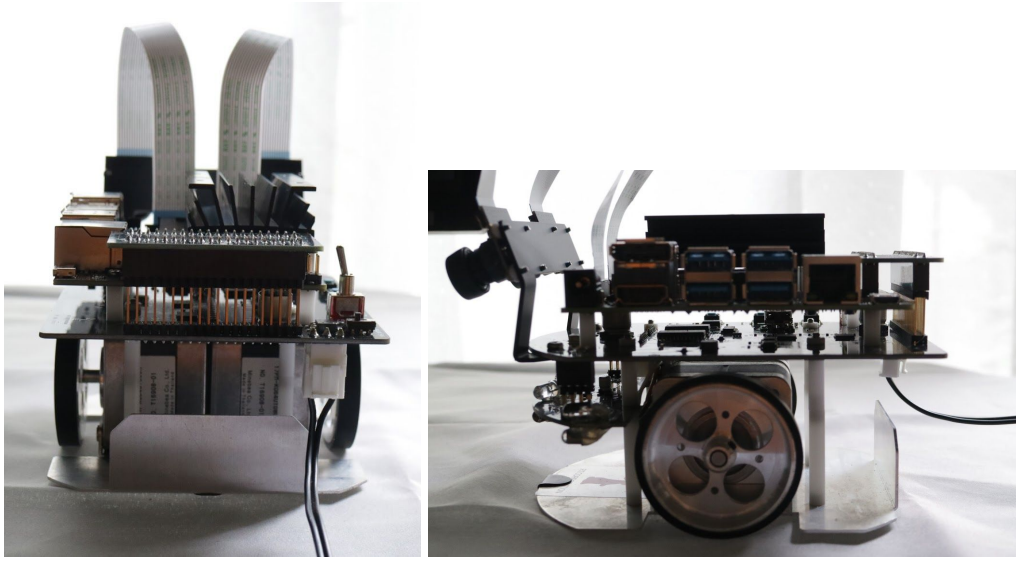

図4.10 変換ケーブルで接続

4.5 開発環境

Jetson NanoにUSBマウスとUSBキーボードとHDMIのモニタを接続します。自動切り替えのHDMIセレクタを使用すると画面が表示されないことがありますので、初期設定を終えるまでは、直接モニタとJetson NanoをHDMIケーブルで接続してください。

図4.11 開発環境

上記の接続例では、Li-Poバッテリの代わりにACアダプタで接続しています。Li-Poバッテリで開発環境を初期設定する際はフル充電したLi-Poバッテリをご使用ください。またインターネットへの接続には別売りのUSB接続の無線LANアダプタを使用しています。

Jetson Nanoを起動して、初期設定をします。上記の状態でMAINと書いてある電源スイッチ(小さい方のスイッチ)をONにします。

しばらくすると下記の画面になります。エンドユーザ使用許諾契約にに同意の上、左下にある「I accept the terms of these license」にチェックを入れて「Continue」のボタンをクリックします。

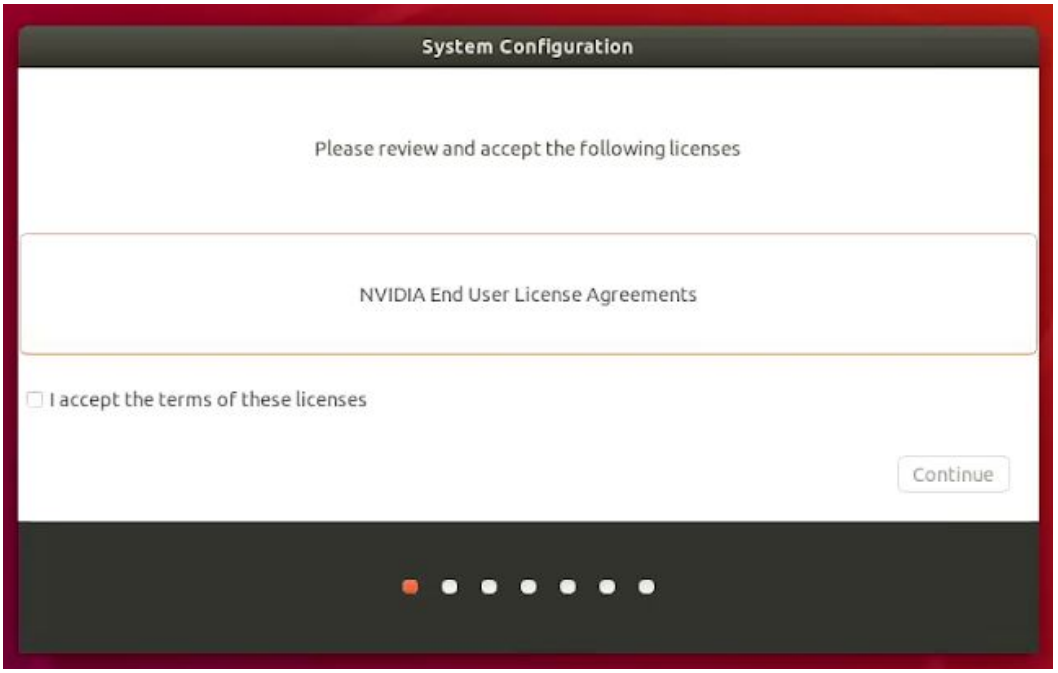

図4.12 ライセンス確認

表示する言語を選択します。ここでは、日本語を選択して、「続ける」のボタンをクリックします。

図4.13 言語選択

次に、使用するキーボードのレイアウトを選択し、キーボード入力を試して確認してから、「続ける」のボタンをクリックします。

図4.14 キーボードレイアウトの選択

時間を設定します。Tokyoであることを確認したら「続ける」のボタンをクリックします。

図4.15 時間設定

ユーザー名とパスワードを設定します。ここでは、ユーザー名とパスワードをubuntuとしています。Wi-Fiを使う方、VNCを使う方は、「自動的にログインする」にチェックをする必要があります。

図4.16 ログインアカウントの設定

OSが使用する容量を設定します。現在のサイズが13GBになっています。特に問題なければ、SDカードの最大容量にします。この例では、最大の30GB(デフォルトが最大容量)にして、「続ける」のボタンをクリックします。

図4.17 パーティションサイズの設定

設定が終わると、下記のログインの待ちの画面になります。

図4.18 ログインアカウント選択画面

リターンキー(enterキー)を押すと下記のパスワードを入れる画面になります。パスワードを入力してリターンキーを押します。

図4.19 パスワード入力

ログインに成功するとショートカットの説明の画面が表示されます。

図4.20 キーボードショートカット

Jetson Nano Mouseでは、SPI、I2Cを使用します。デフォルトでSPI、I2CがイネーブルになっていないバージョンのOSがあるため、SPIとI2Cがイネーブルになっているか確認します。確認にはJetson-IOと呼ばれるツールを使用します。ターミナルに以下のコマンドを入力します。

```
$ sudo /opt/nvidia/jetson-io/jetson-io.py
```

もし起動できない時は、エラー内容によっては対処方法があります。
以下のようなImportErrorが出た場合は、

```
$ sudo /opt/nvidia/jetson-io/jetson-io.py
Traceback (most recent call last):
File "/opt/nvidia/jetson-io/jetson-io.py", line 25, in <module>
from Jetson import board
ImportError: cannot import name 'board'
```

次のコマンドを実行します。

```
$ sudo find /opt/nvidia/jetson-io/ -mindepth 1 -maxdepth 1 -type d -exec
touch {}/__init__.py \;
```

以下のようなRuntimeErrorが出た場合は、

```
$ sudo /opt/nvidia/jetson-io/config-by-pin.py -p 5
Traceback (most recent call last):
```

```
  File "/opt/nvidia/jetson-io/config-by-pin.py", line 51, in <module>
    main()
  File "/opt/nvidia/jetson-io/config-by-pin.py", line 34, in main
    jetson = board.Board()
  File "/opt/nvidia/jetson-io/Jetson/board.py", line 149, in __init__
    self.dtb = _board_get_dtb(self.compat, self.model, dtbdir)
  File "/opt/nvidia/jetson-io/Jetson/board.py", line 88, in
_board_get_dtb
    raise RuntimeError("No DTB found for %s!" % model)
RuntimeError: No DTB found for NVIDIA Jetson Nano Developer Kit!
```

次のコマンドを実行してデバイスツリーのデータをコピーします。

```
$ sudo mkdir -p /boot/dtb
$ ls /boot/*.dtb | xargs -I{} sudo ln -s {} /boot/dtb/
```

修正用のコマンドを実行した後に、もう一度

```
$ sudo /opt/nvidia/jetson-io/jetson-io.py
```

を実行すると下記の画面になります。

図4.21 ピン設定

上図では、19, 21, 23, 24, 26ピンにSPIの機能が割り当てられていないことがわかります。

画面の下にあるカーソルで「Configure 40-pin expandsion header」を選択して、SPIを有効にします。「Configure 40-pin expansion header」を選択してリターンキーを押すと以下の画面になります。

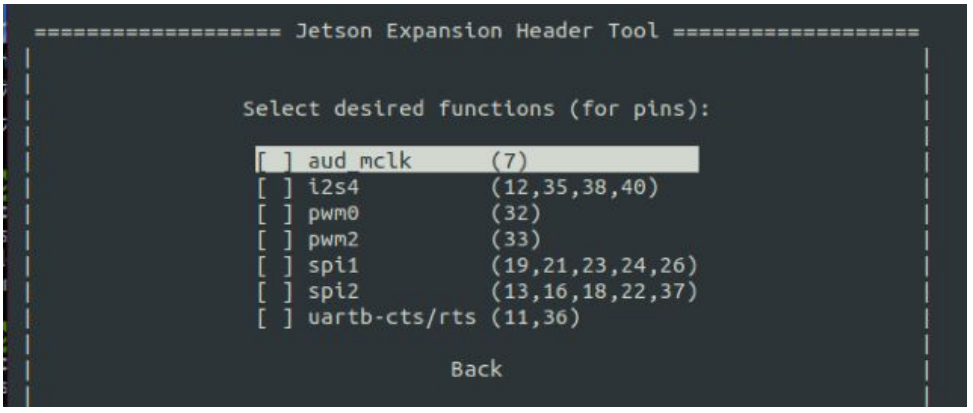

図4.22 ピンファンクション割り当て初期画面

カーソルを「spi1」に持っていき、スペースキーを押すと「 * 」のマークがつきます。

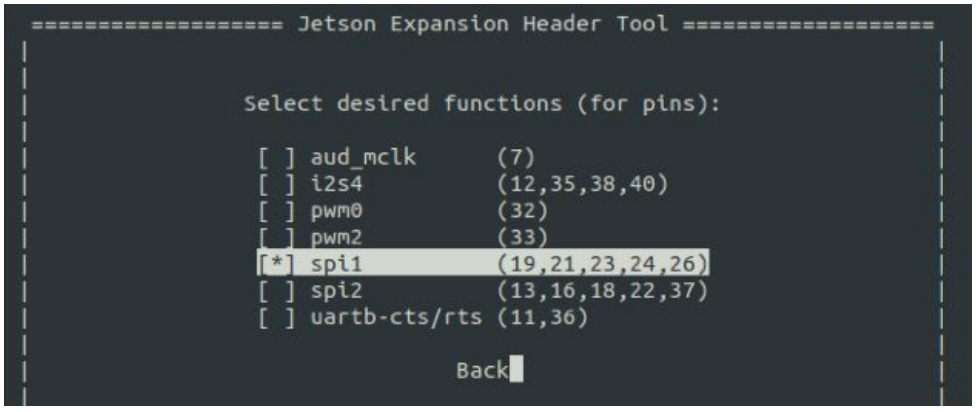

図4.23 ピンファンクションSPIをON

「 * 」が付いていることを確認したら、カーソルを「Back」のところに持っていき、リターンキーを押しspiが設定されていることを確認します。

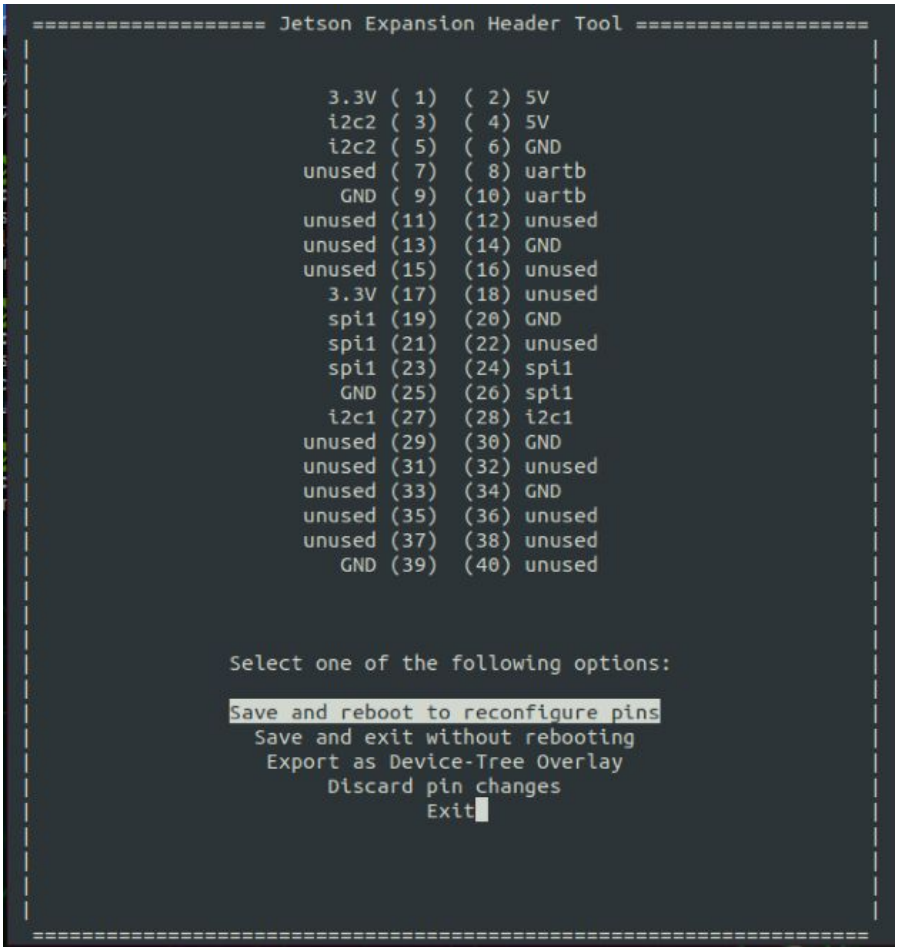

図4.24 ピン設定

19,21,23,24,26にspi1の文字があることが確認できたらカーソルが、「Save and reboot to reconfigure pins」になっていること確認してからリターンキーを押します。成功すると下記の画面になりますので、任意のキーを押して、リブート(再起動)し、設定を反映させます。

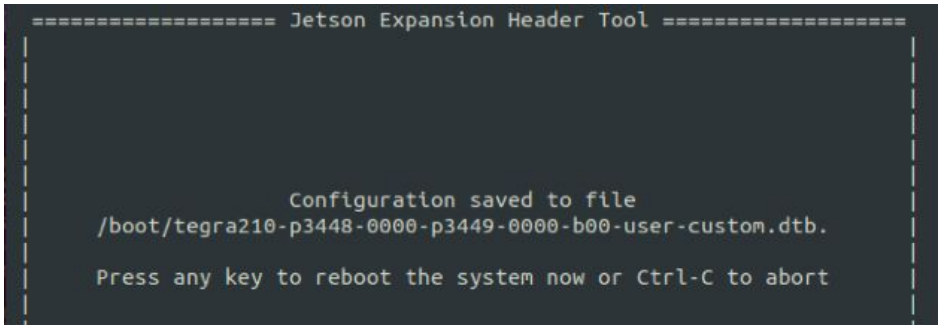

図4.25 ピン設定保存

ターミナル起動は、左上にあるアイコンをクリックし、Searchのところに「term」を入力すると以下の画面になります。「Terminal」をクリックするとコマンドが入力できるターミナルが起動します。

図4.26 ターミナル起動

ショートカットでターミナルを起動する場合は、Ctrl+Alt+Tで起動できます。(VNCなどでMacからログインしているのであれば、control+command+T)

次に、Jetson Nanoと本体をつなぐデバイスドライバをインストールします。デバイスドライバは、GitHubからクローンし、make installする必要があります。

```
$ git clone https://github.com/rt-net/JetsonNanoMouse.git
$ cd JetsonNanoMouse
$ sudo make install
```

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」と言われるもので、機械と入出力をつかさどるファイルです。(/dev/rtc、/dev/rtc0、/dev/rtc1は、JetsonNanoMouseのデバイスファイルではありません)

```
$ ls /dev/rt*
/dev/rtbuzzer0      /dev/rtled3
/dev/rtc            /dev/rtlightsensor0
/dev/rtc0           /dev/rtmotor_raw_l0
/dev/rtc1           /dev/rtmotor_raw_r0
/dev/rtcounter_l0   /dev/rtmotoren0
/dev/rtcounter_r0   /dev/rtswitch0
/dev/rtled0         /dev/rtswitch1
/dev/rtled1         /dev/rtswitch2
/dev/rtled2         /dev/rtswitches0
```

図4.27 デバイスファイル一覧

ドライバをアンインストールする場合は、下記のようにします。

```
$ cd JetsonNanoMouse
$ sudo make uninstall
```

4.5.1 Wi-Fi接続するには

Jetson Nanoには、標準でWi-Fiの機能がついていません。Wi-Fiをつかうには、USB接続の無線LANアダプタをつける必要があります。
Wi-Fiの設定は、右上にある上下の矢印をクリックします。

図4.28 Wi-Fi設定

接続したいWi-Fi Networkをクリックし、パスワードを入力すると接続できます。

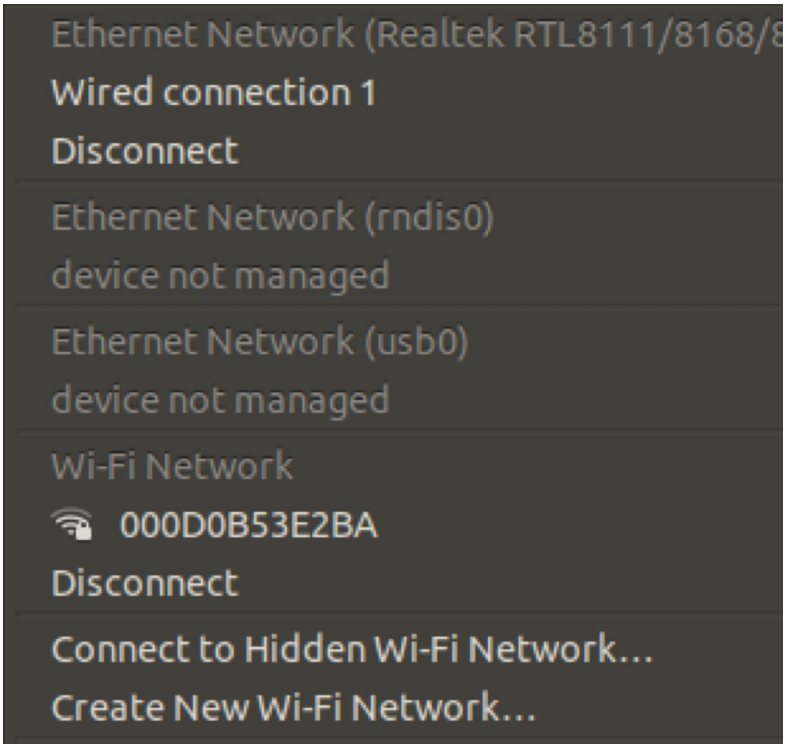

図4.29 Wi-Fi Network

4.5.2 PCからJetson Nanoにリモート接続するには

Jetson NanoにVNCでリモートで接続するには、Jetson NanoをVNCサーバーにする必要があります。VNCサーバーにするとWindows、Ubuntu、macOSからデスクトップ環境に接続できるようになります。

Jetson Nanoにtigervncを入れてVNCサーバーにします。

```
$ sudo apt update
$ sudo apt install tigervnc-standalone-server
$ sudo apt install tigervnc-scraping-server
```

インストールが終わった後VNC接続用のパスワードを設定します。同じパスワードを2回入れる必要があります。最後の質問は「n」とします。

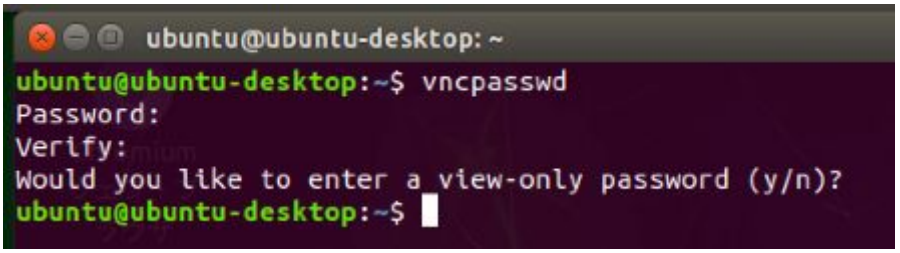

図4.30 VNCパスワード入力

Jetson NanoでVNCサーバーとして起動するには、以下のコマンドを入力します。

```
$ x0vncserver  -passwordfile ~/.vnc/passwd
```

WindowsとmacOSでリモート接続させる方は、RealVNCが使われている方が多い思われますので、RealVNCでの接続を紹介します。
RealVNCを起動し、「<Jetson NanoのIPアドレス>:5900」を入力します。ここの例では、IPは192.168.0.15です。

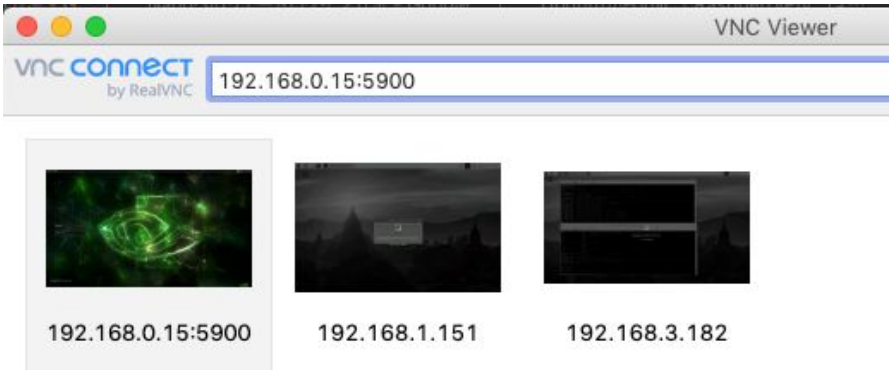

図4.31 RealVNCアドレス入力

リターンキーを押すと下記のメッセージが表示されます。

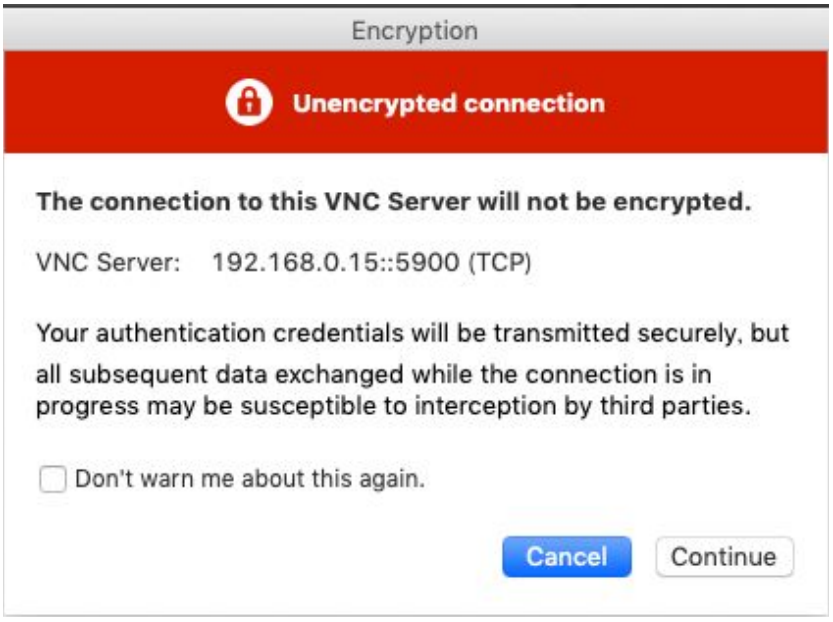

図4.32 RealVNC　注意事項

「Continue」のボタンをクリックし、パスワードを入力します。

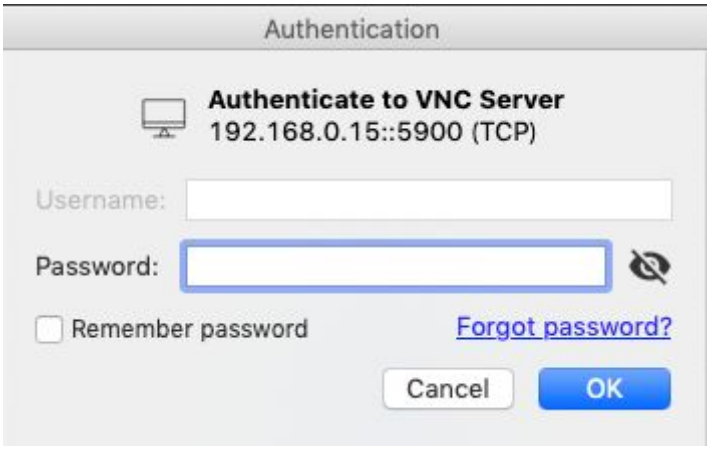

図4.33 RealVNC パスワード入力

Jetson Nanoの画面が表示されると思います。

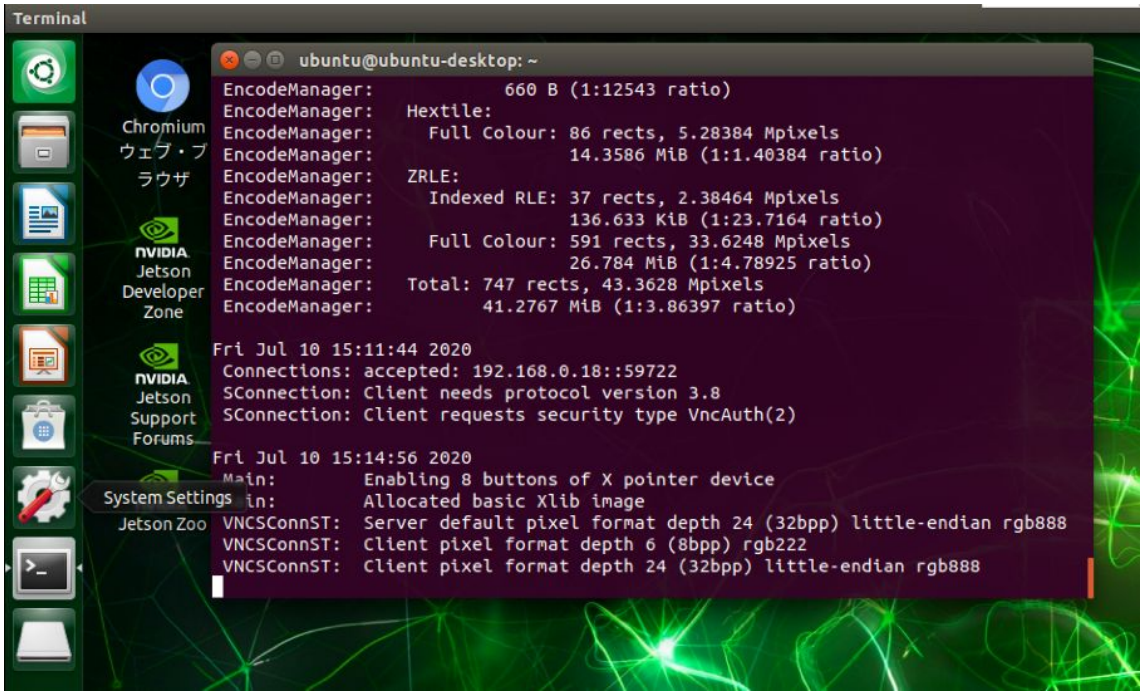

図4.34 画面表示

UbuntuからJetson NanoにVNCでリモートで接続する場合は、標準で入っているRemminaで接続することができます。
Remminaを起動します。

図4.35 Reminaアイコン

左上にある 緑色の「+」 をクリックしてプロファイルを表示します。

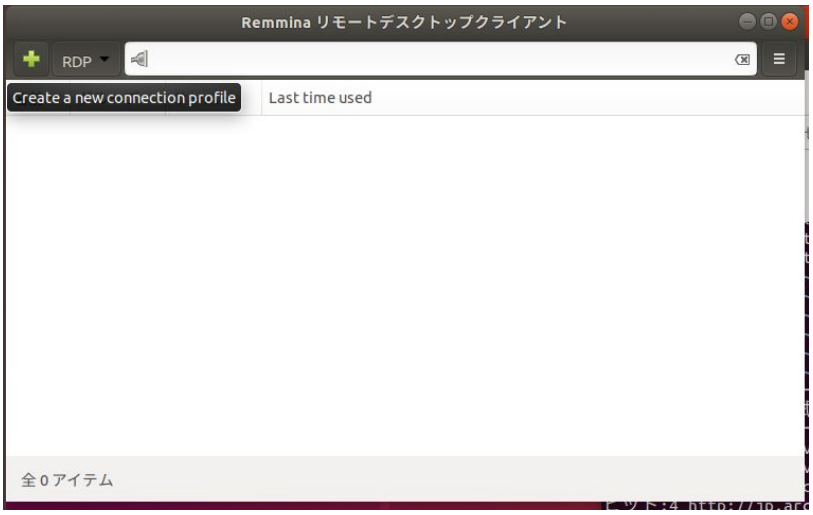

図4.36 Reminaトップ画面

プロトコルをVNCに変更し、サーバーにJetson NanoのIP:5900を入力して、「接続」のボタンをクリックします。この例では、Jetson NanoのIPは192.168.0.15です。

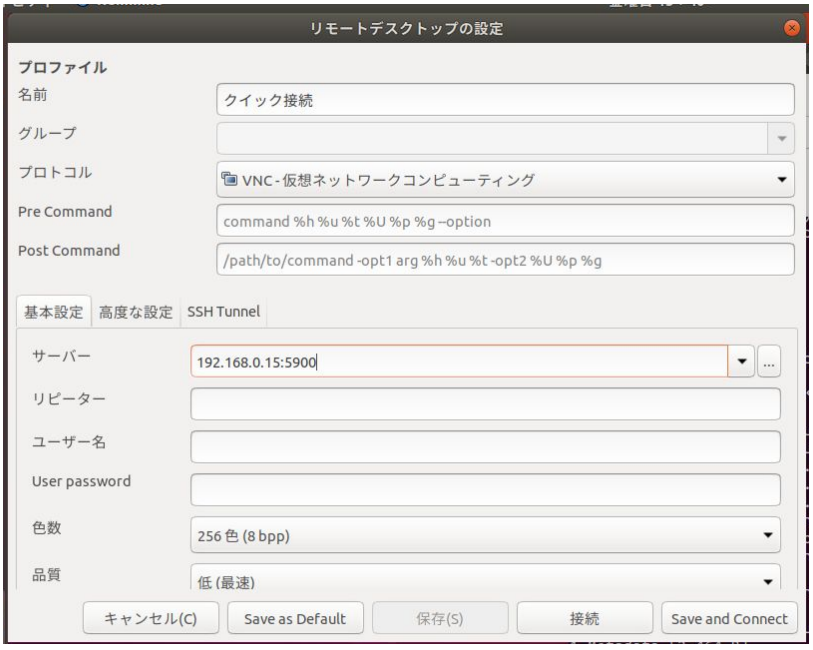

図4.37 Remina接続設定

パスワードを入力して「OK」をクリックすると接続できます。

図4.38 Reminaパスワード入力

Jetson Nanoを起動して、毎回　GUIの画面で “$ x0vncserver -passwordfile ~/.vnc/passwd” でVNCサーバーを起動するのが大変なので、sshでログインした後、下記のコマンドでVNCサーバーを起動することができます。この方法では、自動ログインにしておく必要があります。

```
$ x0vncserver  -display :0 -passwordfile ~/.vnc/passwd
```

ディスプレイを接続せずにJetson Nanoを起動しVNC接続するとフレームバッファが足りず解像度が640x480になってしまう場合があります。解像度が640x480で固定されてしまった場合は、一度モニタをつなぐか、設定ファイルを書き換えて解像度を変更します。ここでの例では、設定ファイルを書き換えて1920x1080にしています。

Jetson Nanoにあるxorg.confを修正します。

```
$ sudo vi /etc/X11/xorg.conf
```

以下をファイルの最後に追加します。

```
Section "Monitor"
   Identifier "DSI-0"
   Option    "Ignore"
EndSection

Section "Screen"
   Identifier    "Default Screen"
   Monitor        "Configured Monitor"
   Device        "Default Device"
   SubSection "Display"
       Depth    24
       Virtual 1920 1080
   EndSubSection
EndSection
```

追加できると以下のスクリーンショットのようになります。

```
ubuntu@ubuntu-desktop: ~
# Copyright (c) 2011-2013 NVIDIA CORPORATION.  All Rights Reserved.

#
# This is the minimal configuration necessary to use the Tegra driver.
# Please refer to the xorg.conf man page for more configuration
# options provided by the X server, including display-related options
# provided by RandR 1.2 and higher.

# Disable extensions not useful on Tegra.
Section "Module"
    Disable     "dri"
    SubSection  "extmod"
        Option  "omit xfree86-dga"
    EndSubSection
EndSection

Section "Device"
    Identifier  "Tegra0"
    Driver      "nvidia"
# Allow X server to be started even if no display devices are connected.
    Option      "AllowEmptyInitialConfiguration" "true"
EndSection

Section "Monitor"
        Identifier "DSI-0"
        Option "Ignore"
EndSection

Section "Screen"
        Identifier "Default Screen"
        Monitor "Configured Monitor"
        Device "Default Device"
        SubSection "Display"
                Depth 24
                Virtual 1920 1080
        EndSubSection
EndSection
~
(END)
```

図4.39 xorg.conf

4.5.3 自動ログインにするには

画面の右上にある歯車状のアイコンをクリックして、「System Settings」を起動します。

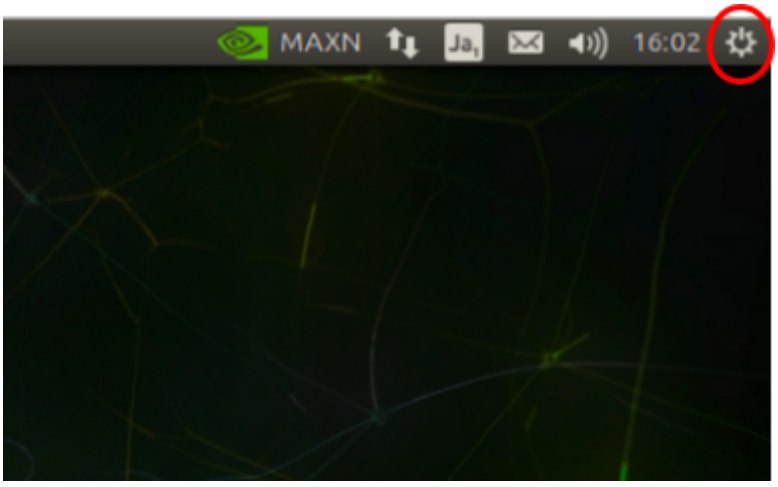

図4.40 ログイン設定

「User Accounts」をクリックします。

図4.41 User Accounts

右上にある「Unlock」のアイコンをクリックして、パスワードを入れた後、「Automatic Login」をOFFからONにすれば、自動ログインになります。

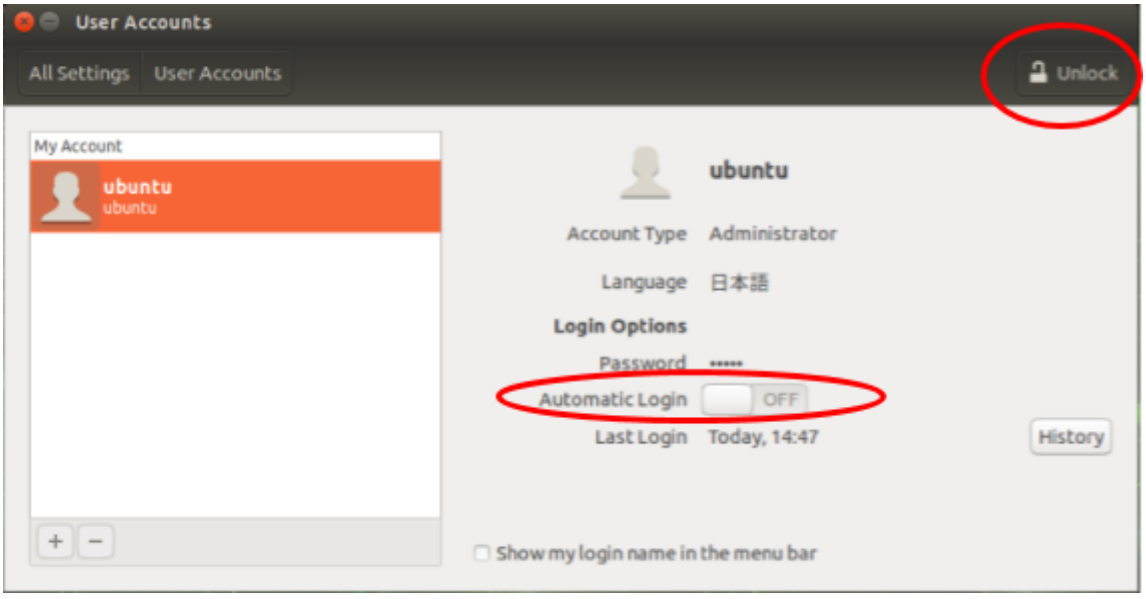

図4.42 Automatic Login ON

5.デバイスドライバ使い方について

デバイスドライバを使うことで、LEDの点灯、ブザーの発信、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、パルスカウントの確認ができます。
コマンドは、RaspberryPiMouseと同じです。ターミナルからコマンドを叩くと実行できます。

5.1 LEDの操作

LEDは0から3まであり、rtledに1を書き込むと点灯し、0を書き込むと消灯します。下記に示すコマンドでLED0が点灯/消灯します。LED1の場合は、rtled1で点灯/消灯ができます。2、3も同様です。

点灯

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

消灯

```
$ echo 0 > /dev/rtled0
```

5.2 ブザーの操作

0と正の整数で周波数(単位:Hz)をrtbuzzer0に書き込むと設定した周波数の音がなります。基板上でブザーへ信号を出力しているICの仕様で一部の周波数については指定しても切り上げられる場合があります。

440Hzで鳴らす

```
$ echo 440 > /dev/rtbuzzer0
```

消音する

```
$ echo 0 > /dev/rtbuzzer0
```

5.3 モータの操作

モータを回すにはもハードウェアスイッチ(MOTOR SW)とソフトスイッチ(/dev/rtmotor0)を両方ONにして初めてモータを回すことができます。プログラミングにミスなくソフトウェアスイッチをON/OFFできるのであれば、ハードウェアスイッチをつける必要はないのですが、JetsonNanoの暴走、プログラム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に、ハードウェアスイッチをつけています。

ソフトウェアスイッチをONにするには、rtmotoren0に1を書き込みます。

```
$ echo 1 > /dev/rtmotoren0
```

左のモータのみを順方向に400Hzで回すにはrtmotor_raw_l0に400を書き込みます。ここでの順方向は、前に進む回転のことを言っています。マイナスの値を入れると逆方向に回ります。/dev/rtmotor_raw_l0と/dev/rtmotor_raw_r0は、**整数のみ入力**できます。基板上でモータへ信号を出力しているICの仕様で一部の周波数については指定しても切り上げられる場合があります。

```
$ echo 400 > /dev/rtmotor_raw_l0
```

左のモータのみを逆方向に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 250 > /dev/rtmotor_raw_r0
$ echo 250 > /dev/rtmotor_raw_l0
```

ソフトウェアスイッチOFF

```
$ echo 0 > /dev/rtmotoren0
```

$echo 0 > /dev/rtmotoren0でもモータの回転は止まりますが、モータを回転を制御するPWMの波形は、出力している状態です。パルスカウンターは、このPWMの波形をカウントしているため、止まっている状態ではあるが、パルスカウントの値が増えていく現象が起きます。パルスカウンタを使う方は、モータの停止には、echo 0 > /dev/rtmotor_raw_l0とecho 0 > /dev/rtmotor_raw_r0で停止するようにしてください。

JetsonNanoMouseに使用されているモータは、ステッピングモータというモータが使われており、モータドライバに1パルスの信号を送ると、0.9[deg]回るモータです。逆に言えば、パルスを送らないと回らないモータです。一回転させるには、400パルス必要です。1秒間に1回転させるには、400という数字をいれます。つまり、回転数に置き換えると1[rps]=400という値になります。

5.4 センサの読み取り

センサの値は、cat などでrtlightsensor0から値を読み出すことができます。JSONのようなkeyとvalueがないので、順番に気をつけてください。

```
$ cat /dev/rtlightsensor0
187 189 184 191
```

左から第1、第2、第3、第4の順番でセンサの値が表示されます。センサの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、開発環境の章で紹介したJetson-IOの設定をもう一度見直してください。

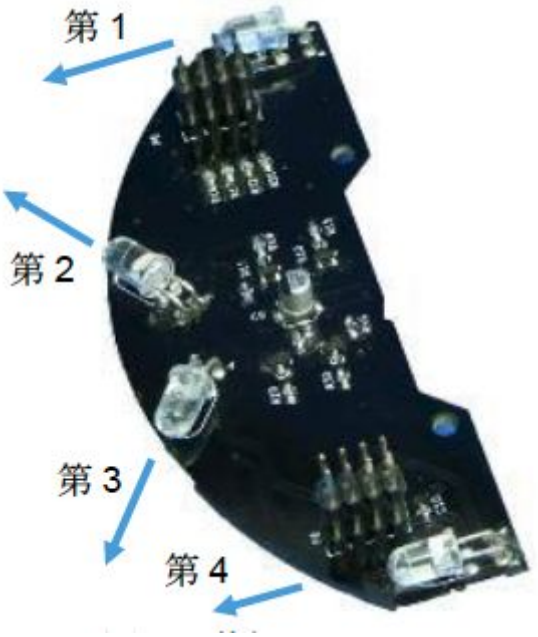

図5.1 光量距離センサ基板

5.5 タクトスイッチの読み取り

タクトスイッチは、SW0からSW2まであり、SWが押されたかどうかを判断します。押されたスイッチは、0として値を返し、押されてない時は、1として値を返します。
下記に示すコマンドの例は、 タクトスイッチSW0が押されたか、押されていないかの結果を示しています。タクトスイッチSW1の場合は、rtswitch1となり、最後の数字を0から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
```

5.6 パルスカウンタの使い方

パルスカウンタのICは、JetsonNanoから出力されたPWMのパルスをカウントし、そのデータをI2Cを使って取り出しています。そのため、I2Cを有効にする必要があります。動作ができないと思われる場合は、開発環境の章で紹介したJetson-IOの設定をもう一度見直してください。

パルスカウンタのrtcounter_l0とrtcounter_r0は読み書きができます。

右側のモータのカウントされているパルス数を確認する場合(読み込み)

```
$ cat /dev/rtcounter_r0
100
```

右側のモータのカウントの値を1000に設定する場合(書き込み)

```
$ echo 1000 > /dev/rtcount_r0
```

書き込んだ結果が出力されないため、書き込まれたかの確認として値の読み込みをします。

```
$ cat /dev/rtcounter_r0
1000
```

5.7 サンプルプログラム

デバイスドライバの簡単な使い方を記述しましたが、ここでは、Shell、C言語も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。 なお、サンプルプログラムは、RaspberryPiMouseと同じであるため、RasberryPiMouseをダウンロード、RaspberryPiMouse/SampleProgramを使用します。

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

5.7.1 STEP1 LEDを光らせよう

step1は表示用のLEDを点滅させます。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。

Shellバージョン

```
#!/bin/bash

while true
do
    echo 1 | tee /dev/rtled?
    sleep 0.5
    echo 0 | tee /dev/rtled?
    sleep 0.5
done

step1.sh (END)
```

図5.2 STEP1 Shell バージョン

動作確認と解説

上記のプログラムがstep1.shにファイルに保存されています。実行権限を追加(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。

中身の解説として、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
```

条件が真の場合(成立している場合、doからdoneまで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。

デバイスファイル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、デバイスドライバのrtled0に “1” が送られ、LED0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(ここでは、デバイスファイル)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。step1.shの例では、 “>”を使わずに “tee”コマンドと “?”(任意の一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までを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せます。 “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。

C言語バージョン

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

int 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]);
    }
    return 0;
}
step1.c (END)
```

図5.3 STEP1 C バージョン

動作確認と解説

上記のプログラムがstep1.cにファイルに保存されています。$ 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。コンパイルで生成された実行ファイルを $ ./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。

中身の解説として、C言語でファイルの読み書きにopen、closeの関数を使用します。ファイルのアクセスに限らず、デバイスファイルのよにも使用できます。open、close関数を使用するには、 “fcntl.h” が必要になり、includeしています。

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)
{
        処理;
}
```

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){ }”とは違い条件式が偽(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){                                   do{
```

```
  write(led0,"1",1);                    write(led0,"1",1);
  usleep(500 * 000);                    usleep(500 * 1000);
  write(led0,"0",1);                    write(led0,"0",1);
  usleep(500 * 1000);                   usleep(500 * 1000);
 }                                     }write(0);
```

LED0が一度も点灯しない　　LED0が一度だけ点滅する

デバイスファイルに出力するコマンドとして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);
```

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 int 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。

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[us]のusleepを使って0.5秒sleepを実現しています、単位は、10^{-3}がms、10^{-6}がusで、us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。

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)
step1.py (END)
```

図5.4 STEP1 Python バージョン

動作確認と解説

上記のプログラムがstep1.pyにファイルに保存されています。$ python step1.pyで実行するか、ファイルに実行権限(chmod +x step1.py)をつけて ./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。

デバイスファイルに値を書き込むコマンドとしてwriteを使用します。write単体で使用するとバッファに入るため、ある程度データを書き込みをしないとデバイスファイルに書いてくれません。withを使うと自動的にclose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する必要があります。
　C言語では、while内で処理する範囲として、{}を使用しますが、Pythonは、{}を使用せず、インデントで制御します。

5.7.2 STEP2 ブザーを鳴らそう

　Jetson Nano Mouseのデバイスドライバを使用すると鳴らしたい周波数をデバイスファイルに書き込むだけで音がなります。キーボードを叩いてピアノのように音階をだ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。

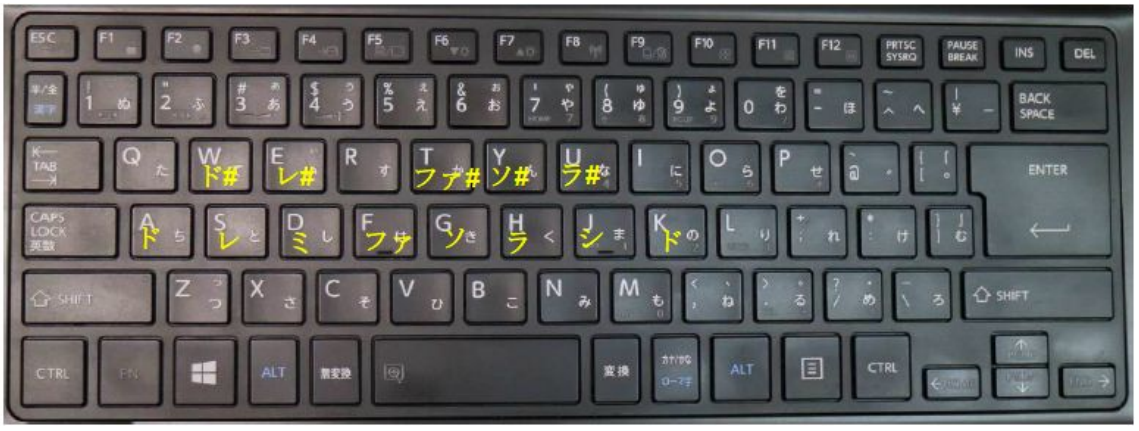

図5.5 音階割り当て

Shellバージョン

```
#!/bin/bash

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

step2.sh (END)
```

図5.6 STEP2 Shell バージョン

```
off 0 0
DO 261 a
DO# 277 w
re 293 s
re# 311 e
mi 329 d
fa 349 f
fa# 370 t
so 392 g
so# 415 y
ra 440 h
ra# 466 u
shi 493 j
do2 523 k

SCALE (END)
```

図5.7 ドレミ周波数

動作確認と解説

上記のプログラムがstep2.shとSCALEのファイルに保存されています。実行権限を追加(chmod +x step2.sh)し、　$ ./step2.sh　で実行し、キーボードを叩くと音が鳴ります。0を押すと消音に鳴ります。終了するには、　Ctrl + c で終了します。

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に代入指定います。　-N　数字のオプションで変数に入れ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。

ハードウェアの仕様で、細かい周波数設定ができません。デバイスドライバの内部で、以下のようにして値を決めています。この計算結果で同じ出力となる場合は、デバイスファイルに書き込んだ周波数が違うにもかかわらず、同じ周波数の出力（同じ音）になります。

計算式

int(25000000 / 4096 / 周波数　+0.5)

C言語バージョン

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

int 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;
            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, "466", 4);
                break;
            case 'j':  // shi
                write(buzzer, "493", 4);
                break;
            case 'k':  // do2
                write(buzzer, "523", 4);
                break;
            case 'c':
                write(buzzer, "0", 2);
                c = 0;
                break;
        }
    }
    close(buzzer);
    return 0;
}
step2.c (END)
```

図5.8 STEP2 C バージョン

動作確認と解説

上記のプログラムがstep2.cにファイルに保存されています。$ gcc step2.c -o step2 でコンパイルします。コンパイルで生成された実行ファイルを $ ./step2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 で終了します。

C言語のgetcharコマンドでは、文字を入力した後リターンキーを叩かないと入力した文字を取得することができません。shellと同じキーを叩いただれで音がなるようにするには、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)、-icanon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(time N)、-icanonが設定されているとき、最小N文字(min N)を設定しています。これにより、1文字入力されたらすぐに変数に反映します。

Pythonバージョン

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


class _Getch:
    def __init__(self):
        pass

    def __call__(self):
        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


freq_dict = {
        "0": "0", "a": "261", "w": "277", "s": "293", "e": "311", "d": "329",
        "f": "349", "t": "370", "g": "392", "y": "415", "h": "440", "u": "466",
        "j": "493", "k": "523"}

while 1:
    getch = _Getch()
    key = getch()
    if key == "c":
        break
    if key in freq_dict:
        with open('/dev/rtbuzzer0', 'w') as f:
            f.write(freq_dict[key])
step2.py (END)
```

図5.9 STEP2 Python バージョン

動作確認と解説

上記のプログラムがstep2.pyにファイルに保存されています。$ python step2.pyで実行するか、ファイルに実行権限(chmod +x step2.py)をつけて ./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で終了します。

Pythonには、連想配列(ディクショナリ)があり、それを使用するとスッキリ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こむ raw_input()　コマンドがありますが、C言語と同様に、リターンキーを叩かないと入力データを取得できません。そこで、

_Getch関数をつくり、キーボードを叩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。

5.7.3 STEP3 スイッチを使おう

スイッチSW0を押すごとにLED3が点灯/消灯を繰り返し、スイッチSW1を押すごとにLED2、LED1が点灯/消灯を繰り返し、スイッチSW2を押すごとにLED0が点灯/消灯を繰り返すプログラムです。

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
        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


step3.sh (END)
```

図5.10 STEP3 Shell バージョン

動作確認と解説

上記のプログラムがstep3.shにファイルに保存されています。実行権限を追加(chmod +x step3.sh)し、　$ ./step3.sh　で実行し、スイッチ(SW0からSW2)を押すとLEDが点灯/消灯します。終了するには、　Ctrl + C で終了します。

タクトスイッチが押されていると、該当するタクトスイッチのデバイスファイルを読み出すと0が返ってきます。タクトスイッチが押されていない状態では、デバイスファイルを読み出すと1が返ってきます。

if grep -q 0 /dev/rtswitch?で真のとき、sleep 0.1で少し待ってから次の処理をするようにしています。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言います。チャタリングの中にチャタリングの状態を調べると “0”と “1”のいずれの値も取れてしまうため、押したのか、押されていないのかわかりません。しかし、チャタリングは、数usから数ms程度で収まるため、sleep 0.1でチャタリングが終わったあと、スイッチの状態を確認しに行っています。

また、shellで四則演算するには、exprが必要になります。

C言語バージョン

```
#include <fcntl.h>
#include <unistd.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];
    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];
}
```

図5.11 STEP3 C バージョン

```
int 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);
            }
        }
        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);
            }
        }
    }
    return 0;
}
(END)
```

図5.12 STEP3 C バージョン

動作確認と解説

　上記のプログラムがstep3.cにファイルに保存されています。$ gcc step3.c -o step3 でコンパイルします。コンパイルで生成された実行ファイルを $ ./step3で実行し、スイッチ(SW0からSW2)を押すとLEDが点灯/消灯します。終了するには、　Ctrl + C で終了します。

/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”で、2つ目は、改行コードです。readで2バイト受信し、１つ目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ので、関数化しています。

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))
step3.py (END)
```

図5.13 STEP3 Python バージョン

動作確認と解説

上記のプログラムがstep3.pyにファイルに保存されています。$ python step3.pyで実行するか、ファイルに実行権限(chmod +x step3.py)をつけて ./step3.pyで実行し、スイッチ(SW0からSW2)を押すとLEDが点灯/消灯します。終了するには、　Ctrl + C で終了します。

Pythonは、数字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。

5.7.4 STEP4 モーターを回そう

その場で旋回します。スペースがない場合は、タイヤを浮かして車輪の回転を確認してください。床に置いて動作を確認する場合は、周りにものがないことを確認して確認をしてください。Jetson Nano Mouseのデバイスドライバではrtmotor0のデバイスファイルが存在しません(2020/9/11時点、今後開発予定)。そのため、一部ファイルの内容を書き換える必要があります。

Shellバージョン

```
#!/bin/bash

MOTOR_EN=/dev/rtmotoren0
MOTOR_R=/dev/rtmotor_raw_r0
MOTOR_L=/dev/rtmotor_raw_l0
MOTOR=/dev/rtmotor0

SPEED=400
TIME_MS=500

echo "Motor On"
echo 1 > $MOTOR_EN
sleep 0.5

echo "Rotate counter-clockwise"
echo -$SPEED > $MOTOR_L
echo $SPEED > $MOTOR_R
sleep 0.5

echo 0 > $MOTOR_L
echo 0 > $MOTOR_R
sleep 0.5

echo "Rotate clockwise"
echo $SPEED > $MOTOR_L
echo -$SPEED > $MOTOR_R
sleep 0.5

echo 0 > $MOTOR_L
echo 0 > $MOTOR_R
sleep 0.5

#echo "Rotate clockwise"
#echo $SPEED -$SPEED $TIME_MS > $MOTOR
#sleep 0.5

#echo "Rotate counter-clockwise"
#echo -$SPEED $SPEED $TIME_MS > $MOTOR

echo "Motor Off"
echo 0 > $MOTOR_EN
step4.sh (END)
```

図5.14 STEP4 Shell バージョン

動作確認と解説

コメントアウトされる前のプログラムがstep4.shにファイルに保存されています。上記と同じように、最後の方にあるところをコメントアウトして確認してください。

実行権限を追加(chmod +x step4.sh)し、　$ ./step4.sh　で実行すると、左に90度旋回し、その後右に90度旋回して終了します。

今までは、直接デバイスファイルに書いていましたが、ここでは、一度変数に置き換えてから、変数に値を書き込むようにしています。

C言語バージョン

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main(void) {
    int motoren = open("/dev/rtmotoren0", O_WRONLY);
    int motor_l = open("/dev/rtmotor_raw_l0", O_WRONLY);
    int motor_r = open("/dev/rtmotor_raw_r0", O_WRONLY);
//    int motor = open("/dev/rtmotor0", O_WRONLY);

    printf("Motor On\n");
    write(motoren, "1", 1);
    usleep(500 * 1000);

    printf("Rotate counter-clockwise\n");
    write(motor_l, "-400", 5);
    write(motor_r, "400", 5);
    usleep(500 * 1000);

    write(motor_l, "0", 5);
    write(motor_r, "0", 5);
    usleep(500 * 1000);

    printf("Rotate clockwise\n");
    write(motor_l, "400", 5);
    write(motor_r, "-400", 5);
    usleep(500 * 1000);

    write(motor_l, "0", 5);
    write(motor_r, "0", 5);
    usleep(500 * 1000);

//    printf("Rotate clockwise\n");
//    write(motor, "400 -400 500", 15);
//    usleep(500 * 1000);
//
//    printf("Rotate counter-clockwise\n");
//    write(motor, "-400 400 500", 15);

    printf("Motor Off\n");
    write(motoren, "0", 1);

    close(motoren);
    close(motor_l);
    close(motor_r);
//    close(motor);
    return 0;
}
step4.c (END)
```

図5.15 STEP4 C バージョン

動作確認と解説

　コメントアウトされる前のプログラムがstep4.cにファイルに保存されています。上記と同じように、motor変数に関わるところをコメントアウトして確認してください。

　$ gcc step4.c -o step4 でコンパイルします。コンパイルで生成された実行ファイルを $ ./step4で実行すると、左に90度旋回し、その後右に90度旋回して終了します。

Pythonバージョン

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

filename_motoren = "/dev/rtmotoren0"
filename_motor_r = "/dev/rtmotor_raw_r0"
filename_motor_l = "/dev/rtmotor_raw_l0"
filename_motor = "/dev/rtmotor0"

SPEED = "400"
TIME_MS = "500"

def motor_drive(freq_l="0", freq_r="0"):
    with open(filename_motor_l, 'w') as f:
        f.write(freq_l)
    with open(filename_motor_r, 'w') as f:
        f.write(freq_r)

print("Motor On")
with open(filename_motoren, 'w') as f:
    f.write("1")
time.sleep(0.5)

print("Rotate counter-clockwise")
motor_drive("-"+SPEED, SPEED)
time.sleep(0.5)

motor_drive("0", "0")
time.sleep(0.5)

print("Rotate clockwise")
motor_drive(SPEED, "-"+SPEED)
time.sleep(0.5)

motor_drive("0", "0")
time.sleep(0.5)

#print("Rotate clockwise")
#with open(filename_motor, 'w') as f:
#    f.write(SPEED+" -"+SPEED+" "+TIME_MS)
#
#time.sleep(0.5)
#
#print("Rotate counter-clockwise")
#with open(filename_motor, 'w') as f:
#    f.write("-"+SPEED+" "+SPEED+" "+TIME_MS)

print("Motor Off")
with open(filename_motoren, 'w') as f:
    f.write("0")
step4.py (END)
```

図5.16 STEP4 Python バージョン

動作確認と解説
　コメントアウトされる前のプログラムがstep4.pyにファイルに保存されています。上記と同じように、最後の方にあるところをコメントアウトして確認してください。
　$ python step4.pyで実行するか、ファイルに実行権限(chmod +x step4..py)をつけて./step4.pyで実行左に90度旋回し、その後右に90度旋回して終了します。
　motor_driveという関数を作って、左右の周波数を一度に入力できるようにしています。

5.7.5 STEP5 センサの値を確認してみよう

　センサの値をひたすらに表示するサンプルプログラムです。センサの前に手をかざしたり、センサにものを近づけたりにすると値が変化することを確認してみましょう。センサの高くにものがあるとセンサの値が大きくなり、遠くなると値が小さくなります。

Shellバージョン

```
#!/bin/bash

while true
do
    cat /dev/rtlightsensor0
    sleep 0.5
done

step5.sh (END)
```

図5.17 STEP5 Shell バージョン

動作確認と解説
　上記のプログラムがstep5.shにファイルに保存されています。実行権限を追加(chmod +x step5.sh)し、　$ ./step5.sh　で実行し、手などをセンサの前にかざすと値が返ってきます。終了するには、　Ctrl + C で終了します。

C言語バージョン

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main(void) {
    int buff_size = 256;
    FILE *fp;

    char buff[buff_size];
    while (1) {
        if ((fp = fopen("/dev/rtlightsensor0", "r")) != NULL) {
            while (fgets(buff, buff_size, fp) != NULL) {
                printf("%s", buff);
            }
        }
        fclose(fp);
        usleep(500 * 1000);
    }
    return 0;
}
step5.c (END)
```

図5.18 STEP5 C バージョン

動作確認と解説
　上記のプログラムがstep5.cにファイルに保存されています。$ gcc step5.c -o step5 でコンパイルします。コンパイルで生成された実行ファイルを $ ./step5で実行し、手などをセンサの前にかざすと値が返ってきます。終了するには、　Ctrl + C で終了します。

Pythonバージョン

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

while 1:
    with open("/dev/rtlightsensor0", 'r') as f:
        print(f.read().strip())
    time.sleep(0.5)

step5.py (END)
```

図5.19 STEP5 Python バージョン

動作確認と解説
　上記のプログラムがstep5.pyにファイルに保存されています。$ python step5.pyで実行するか、ファイルに実行権限(chmod +x step5.py)をつけて ./step5.pyで実行し、、手などをセンサの前にかざすと値が返ってきます。終了するには、　Ctrl + C で終了します。
　f.read().strip()の.strip()は、先頭と末尾にある、文字ではないものを削除しています。ここでは、末尾の改行が削除されています。

5.7.6 STEP6 パルスカウンタの値を確認しよう

　rtmotor_raw_r0に400を与え、1秒後に停止すると、丁度一回転しているはずですが、よく見ると、少しずれていることがあります。これは、OSでは正確な1秒ではなく、1秒近くで停止する動作を行なっているからです。それを補正するため、正確には、どの程度回ったかを確認する手段として、パルスカウンタがあります。モータードライバに与えたパルスをカウントしているので正確な回転数を知ることができます。そのパルスカウンタの動作確認として、左右のモータを回し、カウントを表示するサンプルプログラムです。数秒間モータが回るので、タイヤを浮かせるか、モータの物理スイッチをOffにして確認するようにしてください。

Shellバージョン

```
#!/bin/bash

MOTOR_EN=/dev/rtmotoren0
MOTOR_R=/dev/rtmotor_raw_r0
MOTOR_L=/dev/rtmotor_raw_l0
COUNTER_R=/dev/rtcounter_r0
COUNTER_L=/dev/rtcounter_l0

SPEED=400

function echo_counter () {
    SECONDS=0
    while [ $SECONDS -lt "$1" ]
    do
    echo "count_l:$(cat $COUNTER_L), count_r:$(cat $COUNTER_R)"
    done
}

function reset_counters_and_motors () {
    echo 0 | tee $MOTOR_L $MOTOR_R > /dev/null
    echo "Reset counter"
    echo 0 | tee $COUNTER_L $COUNTER_R > /dev/null
    echo "count_l:$(cat $COUNTER_L), count_r:$(cat $COUNTER_R)"
}

reset_counters_and_motors

echo "Motor On"
echo 1 > $MOTOR_EN

echo "Rotate left motor"
sleep 0.5
echo $SPEED > $MOTOR_L
echo_counter 2
reset_counters_and_motors

echo "Rotate right motor"
sleep 0.5
echo $SPEED > $MOTOR_R
echo_counter 2
reset_counters_and_motors

echo "Move forward"
sleep 0.5
echo $SPEED | tee $MOTOR_L $MOTOR_R > /dev/null
echo_counter 2
reset_counters_and_motors

echo "Move backward"
sleep 0.5
echo -$SPEED | tee $MOTOR_L $MOTOR_R > /dev/null
echo_counter 2
reset_counters_and_motors

echo "Motor Off"
echo 0 > $MOTOR_EN
step6.sh (END)
```

図5.20 STEP6 Shell バージョン

動作確認と解説

　上記のプログラムがstep6.shにファイルに保存されています。実行権限を追加(chmod +x step6.sh)し、　$ ./step6.sh　で実行すると、左のモータだけが回り、その後、右のモータだけが回ります。その後、前進の方向に左右のモータが回り、最後に後退方向に左右のモータが回ります。パルスをカウントしているだけなので、回転方向は検出できていません。

　2秒ごとにモータを回す方法を変えていますが、sleepを使ってしまうと、sleepしている間のパルスカウンタの値を確認することができません。ここでは、2秒という時間を、bashの特殊変数のSECONDSを使って計測しています。このSECONDS変数を参照するとbashが起動されてからの秒数が取得でき、任意のタイミングで値を代入することによってリセットすることができます。モータ止めたり、カウンタの値を確認する処理を関数化してあります。

C言語バージョン

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#define FILE_MOTOREN "/dev/rtmotoren0"
#define FILE_MOTOR_L "/dev/rtmotor_raw_l0"
#define FILE_MOTOR_R "/dev/rtmotor_raw_r0"
#define FILE_COUNT_L "/dev/rtcounter_l0"
#define FILE_COUNT_R "/dev/rtcounter_r0"
#define BUFF_SIZE 256

void motor_drive(char *freq_l, char *freq_r) {
    FILE *motor_l, *motor_r;
    if ((motor_l = fopen(FILE_MOTOR_L, "w")) != NULL &&
        (motor_r = fopen(FILE_MOTOR_R, "w")) != NULL) {
        fputs(freq_l, motor_l);
        fputs(freq_r, motor_r);
    }
    fclose(motor_l);
    fclose(motor_r);
}

void delete_newline(char *str) {
    char *p;

    if ((p = strchr(str, '\n')) != NULL) {
        *p = '\0';
    }
}

void print_counter(const int timeout) {
    FILE *count_l, *count_r;
    char buff_l[BUFF_SIZE];
    char buff_r[BUFF_SIZE];

    time_t start = time(NULL);
    int elapsed_time = 0;

    while (elapsed_time < timeout) {
        if ((count_l = fopen(FILE_COUNT_L, "r")) != NULL &&
            (count_r = fopen(FILE_COUNT_R, "r")) != NULL) {
            while (fgets(buff_l, BUFF_SIZE, count_l) != NULL) {
            }
            while (fgets(buff_r, BUFF_SIZE, count_r) != NULL) {
            }
            delete_newline(buff_l);
            delete_newline(buff_r);
            printf("count_l:%s, count_r:%s\n", buff_l, buff_r);
        }
        fclose(count_l);
        fclose(count_r);

        elapsed_time = (int)(time(NULL) - start);
    }
}
```

図5.21 STEP6 C バージョン

```
void reset_counters_and_motors(void) {
    FILE *count_l, *count_r;

    motor_drive("0", "0");

    printf("Reset counter\n");
    if ((count_l = fopen(FILE_COUNT_L, "w")) != NULL &&
        (count_r = fopen(FILE_COUNT_R, "w")) != NULL) {
        fputs("0", count_l);
        fputs("0", count_r);
    }
    fclose(count_l);
    fclose(count_r);
}

int main(void) {
    int motoren = open("/dev/rtmotoren0", O_WRONLY);
    // int motor_l = open("/dev/rtmotor_raw_l0",O_WRONLY);

    printf("Motor On\n");
    write(motoren, "1", 1);

    printf("Rotate left motor\n");
    usleep(500 * 1000);
    motor_drive("400", "0");
    print_counter(2);
    reset_counters_and_motors();

    printf("Rotate right motor\n");
    usleep(500 * 1000);
    motor_drive("0", "400");
    print_counter(2);
    reset_counters_and_motors();

    printf("Move forward\n");
    usleep(500 * 1000);
    motor_drive("400", "400");
    print_counter(2);
    reset_counters_and_motors();

    printf("Move backward\n");
    usleep(500 * 1000);
    motor_drive("-400", "-400");
    print_counter(2);
    reset_counters_and_motors();

    printf("Motor Off\n");
    write(motoren, "0", 1);

    close(motoren);
    return 0;
}
(END)
```

図5.22 STEP6 C バージョン

動作確認と解説

上記のプログラムがstep6.cにファイルに保存されています。$ gcc step6.c -o step6 でコンパイルします。コンパイルで生成された実行ファイルを $ ./step6で実行すると、左のモータだけが回り、その後、右のモータだけが回ります。その後、前進の方向に左右のモータが回り、最後に後退方向に左右のモータが回ります。

表示をみやすく、shellの時に同じするため、delete_newlineの関数で改行をヌル文字(¥0)に書き換えて改行を阻止しています。

Pythonバージョン

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

filename_motoren = "/dev/rtmotoren0"
filename_motor_r = "/dev/rtmotor_raw_r0"
filename_motor_l = "/dev/rtmotor_raw_l0"
filename_motor = "/dev/rtmotor0"
filename_count_r = "/dev/rtcounter_r0"
filename_count_l = "/dev/rtcounter_l0"

SPEED = "400"


def motor_drive(freq_l="0", freq_r="0"):
    with open(filename_motor_l, 'w') as f:
        f.write(freq_l)
    with open(filename_motor_r, 'w') as f:
        f.write(freq_r)


def print_counter(timeout=2.0):
    start = time.time()
    while time.time() - start < timeout:
        count_r = count_l = 0
        with open(filename_count_l, 'r') as f:
            count_l = f.read().strip()
        with open(filename_count_r, 'r') as f:
            count_r = f.read().strip()
        print("count_l:" + count_l + ", count_r:" + count_r)


def reset_counters_and_motors():
    motor_drive("0", "0")
    print("Reset counter")
    with open(filename_count_l, 'w') as f:
        f.write("0")
    with open(filename_count_r, 'w') as f:
        f.write("0")
    print_counter(0)
```

図5.23 STEP6 Python バージョン

```
print("Motor On")
with open(filename_motoren, 'w') as f:
    f.write("1")

print("Rotate left motor")
time.sleep(0.5)
motor_drive(SPEED, "0")
print_counter(2.0)
reset_counters_and_motors()

print("Rotate right motor")
time.sleep(0.5)
motor_drive("0", SPEED)
print_counter(2.0)
reset_counters_and_motors()

print("Move forward")
time.sleep(0.5)
motor_drive(SPEED, SPEED)
print_counter(2.0)
reset_counters_and_motors()

print("Move backward")
time.sleep(0.5)
motor_drive("-"+SPEED, "-"+SPEED)
print_counter(2.0)
reset_counters_and_motors()

print("Motor Off")
with open(filename_motoren, 'w') as f:
    f.write("0")
(END)
```

図5.24 STEP6 Python バージョン

動作確認と解説

　上記のプログラムがstep6.pyにファイルに保存されています。$ python step6.pyで実行するか、ファイルに実行権限(chmod +x step6.py)をつけて ./step6.pyで実行すると、左のモータだけが回り、その後、右のモータだけが回ります。その後、前進の方向に左右のモータが回り、最後に後退方向に左右のモータが回ります。

6.お問い合わせ

If you have any inquiries upon this product, please contact us at the following.

RT Corporation　株式会社アールティ
住所：　〒101-0021　東京都千代田区外神田3-2-13山口ビル3F
Address:　3F, 3-2-13 Sotokanda, Chiyodaku 101-0021, Tokyo, Japan
TEL +81-3-6666-2566 FAX +81-3-5809-5738
E-mail: shop@rt-net.jp
Open: 11:00a.m.- 18:00p.m. (JST+9)
Close: weekend, national holiday, summer vacation, new year

改版履歴

Date (YY/MM/DD)	Version	Reference	Editor
2020/09/14	1.0版	Release	中川(範)
2020/09/14	0.6版	チェック、用語修正	中川(範)
2020/09/14	0.5版	画像差し替え、表記ゆれ修正	佐藤(大)
2020/09/11	0.4版	レイアウト修正	中川(範)
2020/09/11	0.3版	JetPack等最新の情報に更新	佐藤(大)
2020/08/17	0.2版	全体内容更新	青木m
2020/06/04	0.1版	作成	小笹