TurtleReal

ー Raspberry PiとROSで動かす安価な2軸駆動ロボット ー

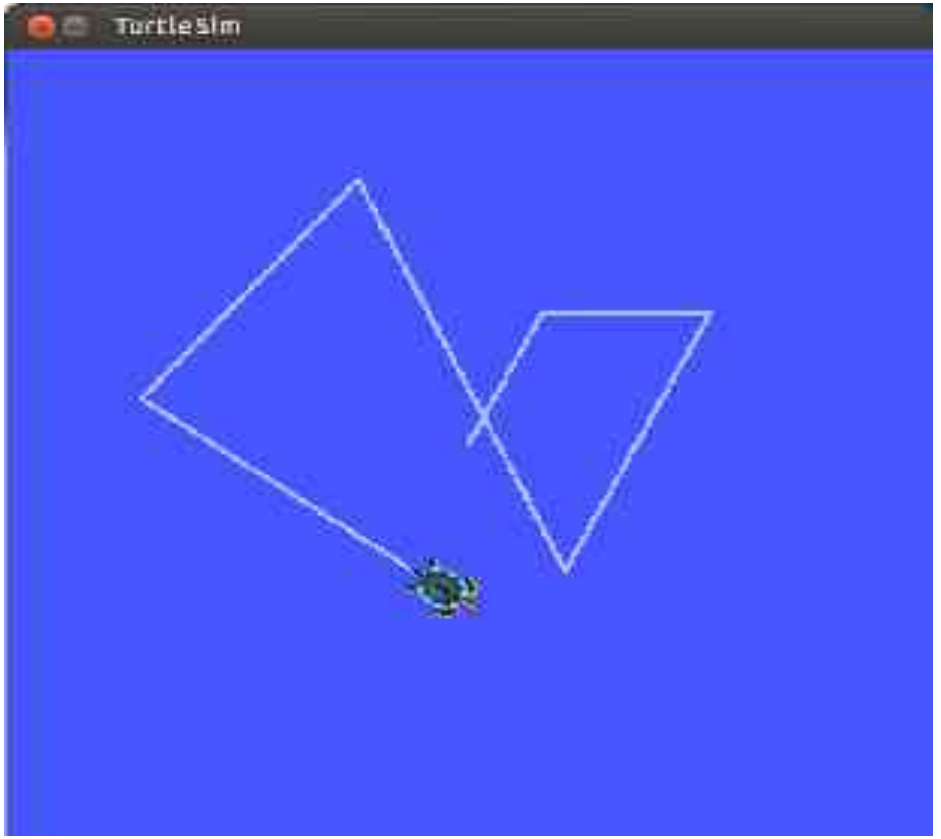
第五回ROS勉強会@名古屋 2015/4/11

なぜTurtleReal？ - 動機と名前の由来 -

- ROSで動く安価なロボットを作りたい。
- TurtleSimの実機版なのでTurtleReal

TurtleSimって何？

TurtleSimはROSのチュートリアルでよく使われる
ウミガメロボットのシミュレーターです。



TurtleSimデモンストレーション

TurtleSimのデモンストレーション

TurtleRealで学べる3つの技術的ポイント

(1) Raspberry PiでのROS利用

(2) 異種マシン間でROSを協調させて動かす

(3) パッケージの使い回し
（TurtleSimの操作ソフトを流用）

TurtleRealで学べる3つの技術的ポイント (1)

(1) Raspberry PiでのROS利用

(2) 異種マシン間でROSを協調させて動かす

(3) パッケージの使い回し
（TurtleSimの操作ソフトを流用）

Raspberry Piって何？

Raspberry Piは、Raspberry Pi財団 によって開発されたARMプロセッサを載せた格安の教育用シングルボードコンピュータでOSとしてRaspberry Pi版Debian(Raspbian)が動作します。

Model A $25 (2013)

Model A+ $20 (2014)

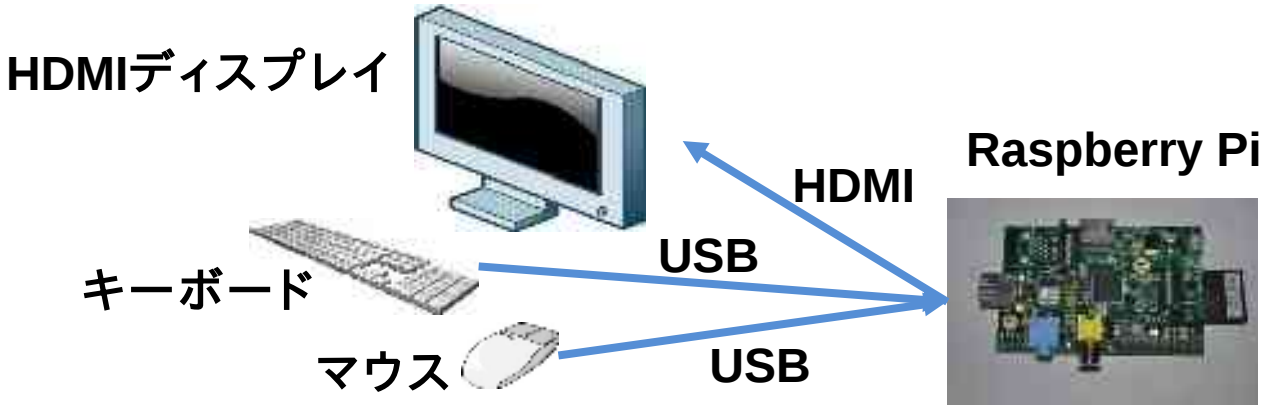
Model B $35 (2012)

Model B+ $35 (2014)

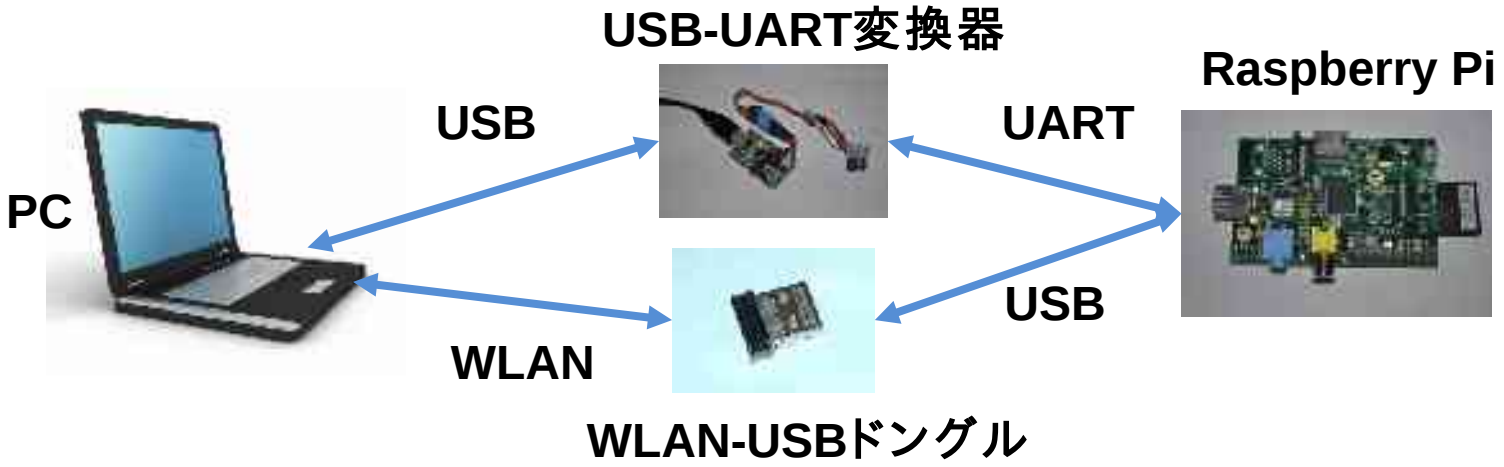
2 Model B $35 (2015)

Raspberry Piの操作環境

(1) PC的な操作環境（GUI）



(2) 組み込み的な操作環境（CUI）



Raspberry PiへのROSインストール

■ROS（Hydro)の場合、バイナリーパッケージがないのでソースからのビルドになりますので結構時間がかかります。手順も複雑なので詳細は以下のリンクを参照ください。

http://forestofazumino.web.fc2.com/ros/ros_install.html

■最新のRaspberry Pi 2 Model Bの場合はUbuntu14.04とUbutuで動作確認されたROS(Indigo)のバイナリーパッケージがありますのでインストールは容易です。

TurtleRealで学べる3つの技術的ポイント (2)

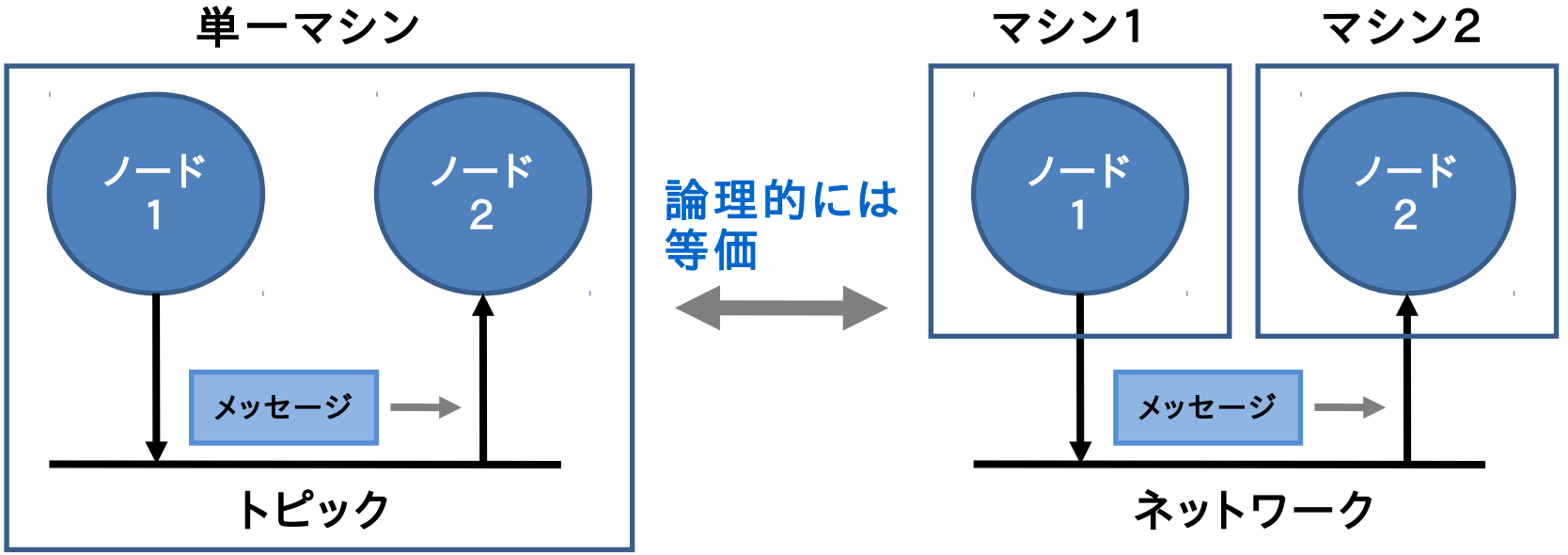
(1) Raspberry PiでのROS利用

(2) 異種マシン間でROSを協調させて動かす

(3) パッケージの使い回し
（TurtleSimの操作ソフトを流用）

複数マシン間のメッセージ通信

ROSではネットワークで繋がれたマシンのどこにノードがあってもメッセージを送ることが出来ますので、遠隔操作、モニタリング、負荷分散などが簡単に出来ます。



Raspberry PiとノートPCをつなぐ

(1) 各マシンの/etc/hostsに以下例のようにIPアドレスとホスト名を記載します。

```
192.168.1.10 NotePC
192.168.1.11 RaspberryPi
```

(2) ノートPC側をマスターにする場合、各マシンで以下のコマンドで環境変数ROS_MASTER_URIを設定します。

```
$ export ROS_MASTER_URI=http://NotePC:11311
```

(3) 以下のコマンドでノートPCでマスターを起動します。

```
$ roscore
```

ネットワークが正常に繋がって入れば、以上で2台のマシン間が繋がります。

TurtleRealで学べる3つの技術的ポイント (3)

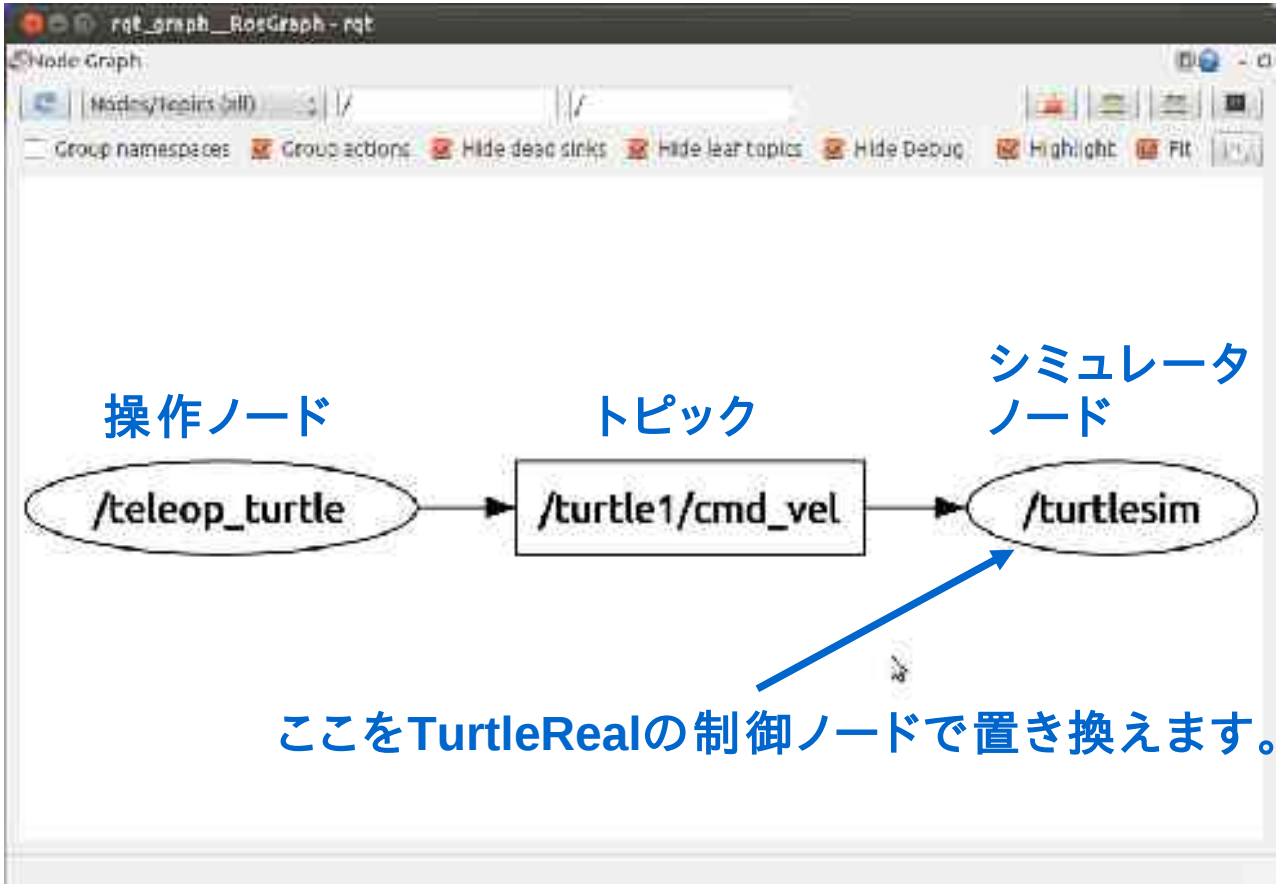
(1) Raspberry PiでのROS利用

(2) 異種マシン間でROSを協調させて動かす

(3) パッケージの使い回し
（TurtleSimの操作ソフトを流用）

TurtleSimのグラフ図

TurtleSimのグラフ（ノード、トピックの接続状態）を
rqt_graphで可視化すると以下のようになります。



メッセージ構造

TurtleSimで使っている位置・姿勢指令のメッセージ構造

```
geometry_msgs/Twist    ← メッセージ名
  geometry_msgs/Vector3 linear
    float64 x    ← X位置
    float64 y    ← Y位置
    float64 z    ← Z位置（今回未使用）
  geometry_msgs/Vector3 angular
    float64 x    ← X軸回りの回転角（今回未使用）
    float64 y    ← Y軸回りの回転角（今回未使用）
    float64 z    ← Z軸回りの回転角
```

TurtleRealにロボットとして必要な機能

(1) 車輪で位置・姿勢を変える

(2) 位置・姿勢検出

(3) 車輪駆動用モーター制御回路

(4) Raspberry Piとモーター制御回路のインターフェース

(5) バッテリー駆動

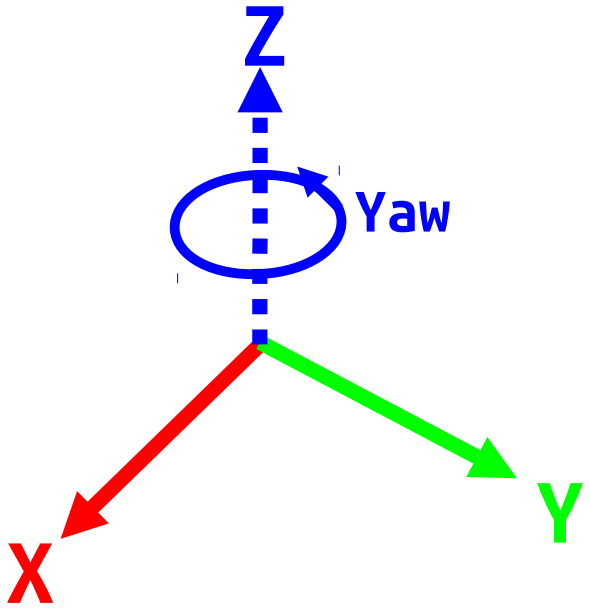
(6) メッセージを指令値とする位置・姿勢制御

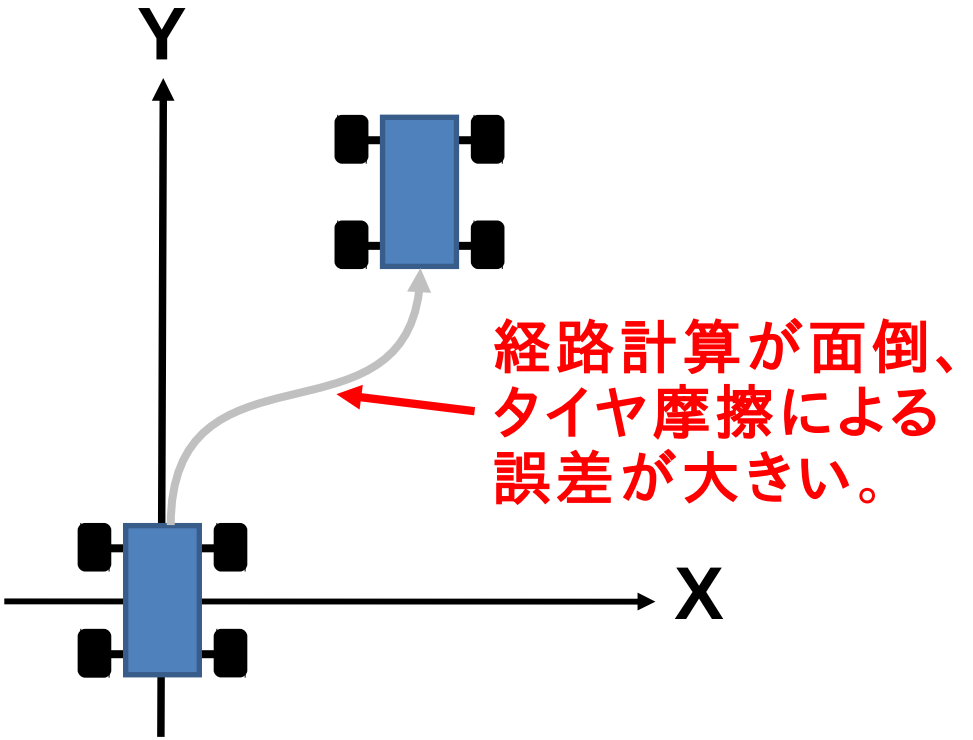
機能の実現手段 (1)

(1) 車輪で位置・姿勢を変える
→2軸車輪による移動
(2) 位置・姿勢検出
→USB光学マウス
(3) 車輪駆動用モーター制御回路
→Hブリッジ回路でPWM制御
(4) Raspberry Piとモーター制御回路のインターフェース
→Raspberry PiのGPIO直結
(5) バッテリー駆動
→モーター電源は6Vバッテリー直結
→その他は低損失の5Vレギュレーターを使用
(6) メッセージを指令値とする位置・姿勢制御
→位置計算ノード
→位置制御ノード

二次元平面による位置･姿勢の変更

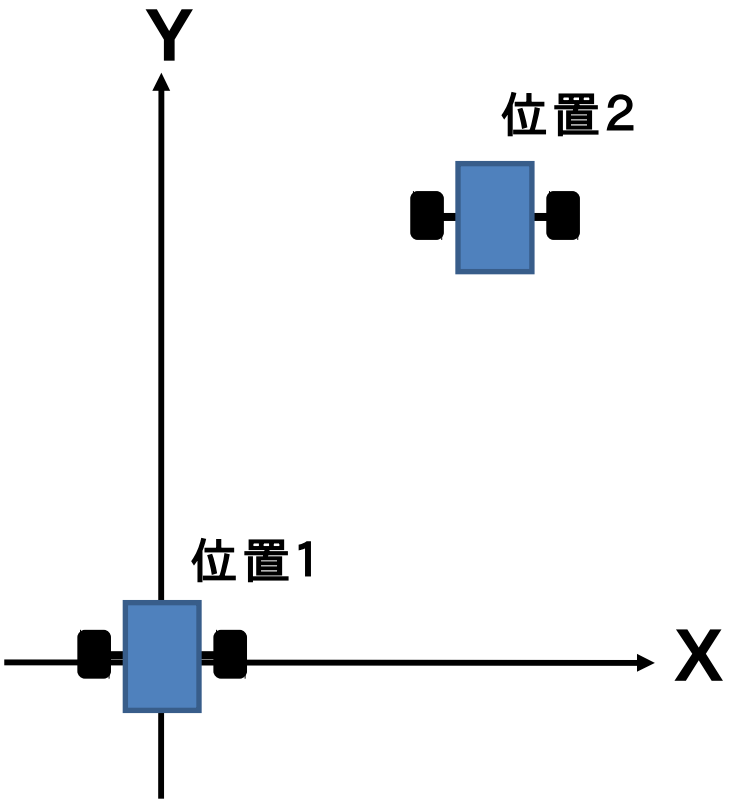
二次元平面で最短かつ同時に位置･姿勢を変えるには3自由度(X, Y, Yaw)が必要です。
自動車（2自由度）の場合、姿勢変更のために距離を長くして経路を調整することで姿勢を変えています。

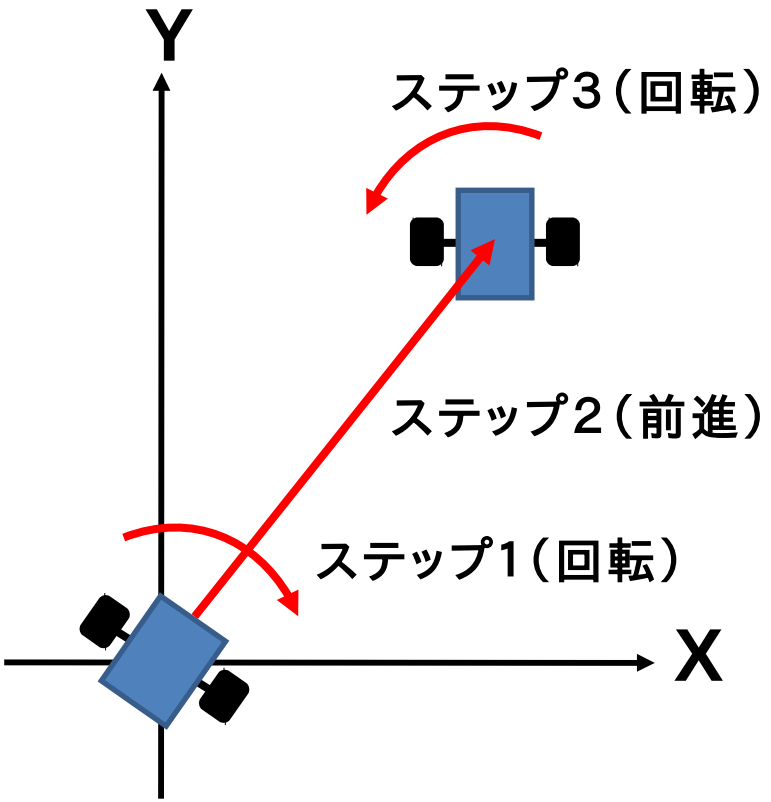




2軸車輪による位置・姿勢の変更

2軸車輪によるロボットは車輪2軸を逆回転させることでYaw回転を実現します。従って位置・姿勢制御は以下の3ステップで行いますが制御は簡単になります。





機能の実現手段 (2)

(1) 車輪で位置・姿勢を変える
→2軸車輪による移動

(2) 位置・姿勢検出
→USB光学マウス

(3) 車輪駆動用モーター制御回路
→Hブリッジ回路でPWM制御

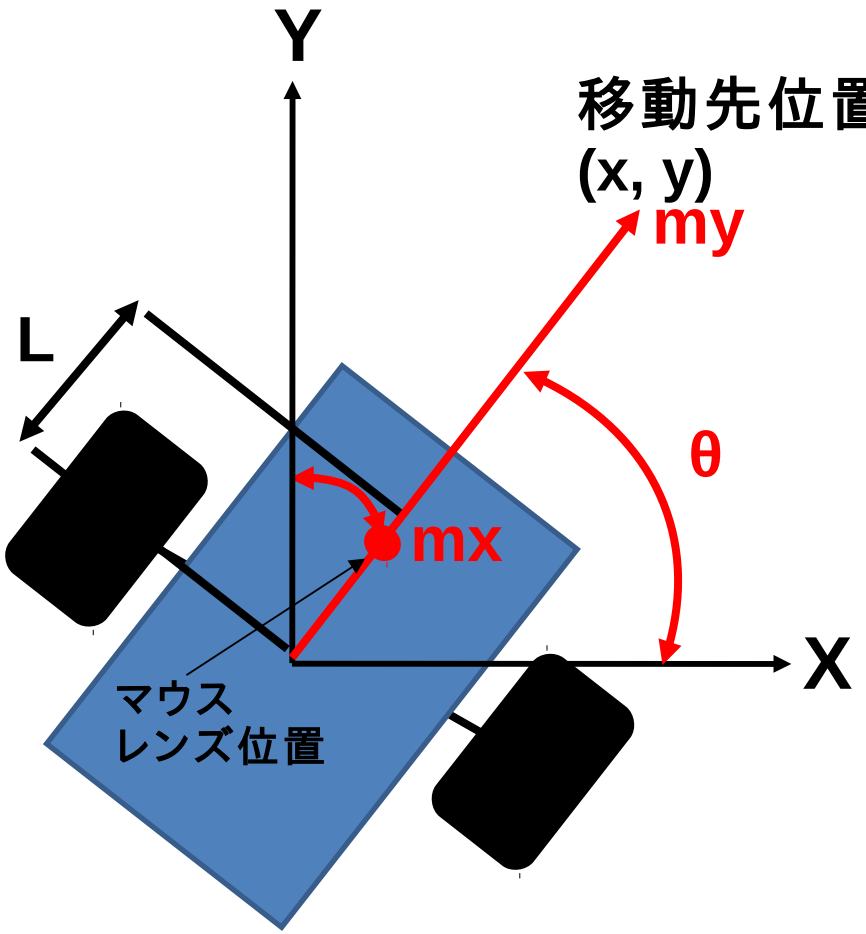
(4) Raspberry Piとモーター制御回路のインターフェース
→Raspberry PiのGPIO直結

(5) バッテリー駆動
→モーター電源は6Vバッテリー直結
→その他は低損失の5Vレギュレーターを使用

(6) メッセージを指令値とする位置・姿勢制御
→位置計算ノード
→位置制御ノード

マウスによる位置･姿勢計算

位置･姿勢は車軸から離れた場所に配置した光学マウスの移動量mx, myから以下のように計算します。



$$\theta = \pi / 2 - mx / L \ [rad]$$
$$x = my \times \cos(\theta)$$
$$y = my \times \sin(\theta)$$

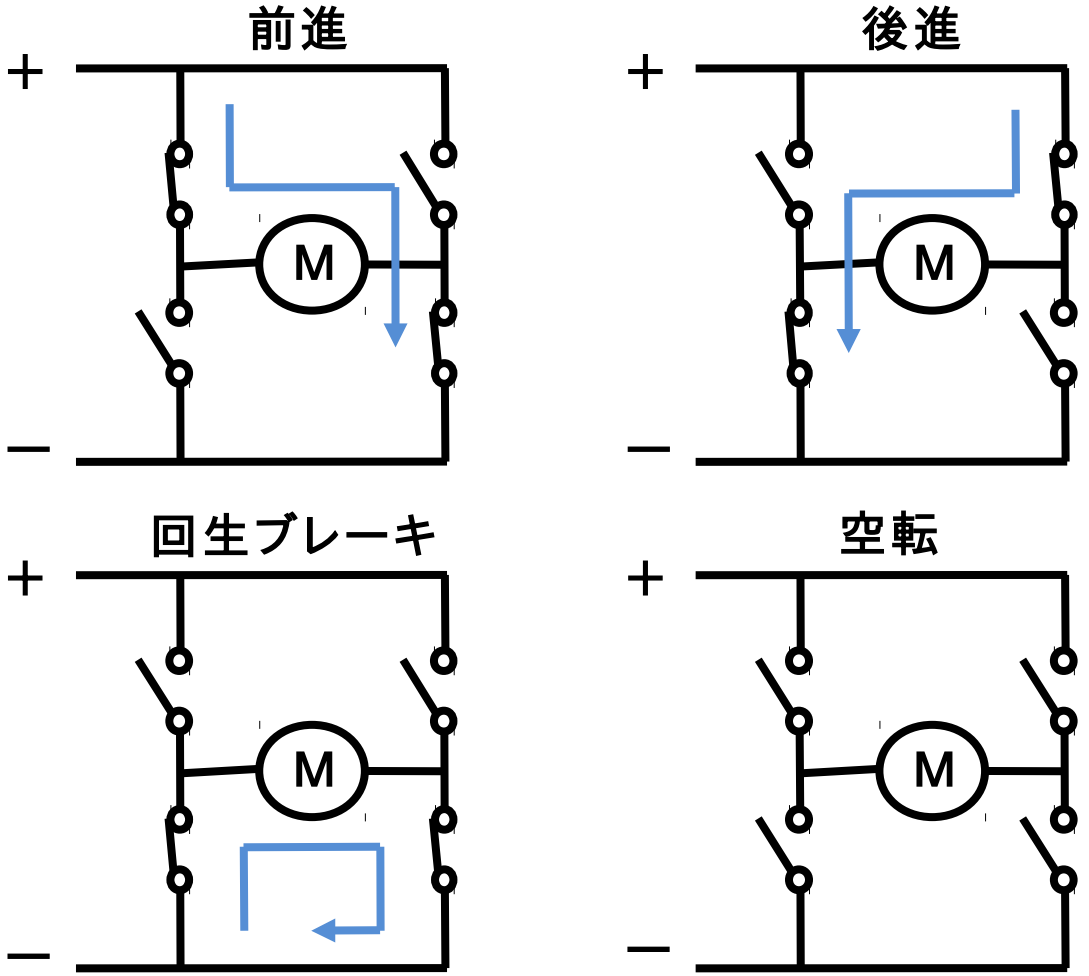
mx: マウスX方向移動量
my: マウスY方向移動量
L: マウスレンズ･車軸間距離
θ: ロボット回転角

機能の実現手段 (3)

(1) 車輪で位置・姿勢を変える
→2軸車輪による移動
(2) 位置・姿勢検出
→USB光学マウス
(3) 車輪駆動用モーター制御回路
→Hブリッジ回路でPWM制御
(4) Raspberry Piとモーター制御回路のインターフェース
→Raspberry PiのGPIO直結
(5) バッテリー駆動
→モーター電源は6Vバッテリー直結
→その他は低損失の5Vレギュレーターを使用
(6) メッセージを指令値とする位置・姿勢制御
→位置計算ノード
→位置制御ノード

Hブリッジ回路によるモーター駆動

Hブリッジ回路でモーターの前進、後進、ブレーキ制御をします。



機能の実現手段 (4)(5)

(1) 車輪で位置・姿勢を変える
→2軸車輪による移動
(2) 位置・姿勢検出
→USB光学マウス
(3) 車輪駆動用モーター制御回路
→Hブリッジ回路でPWM制御

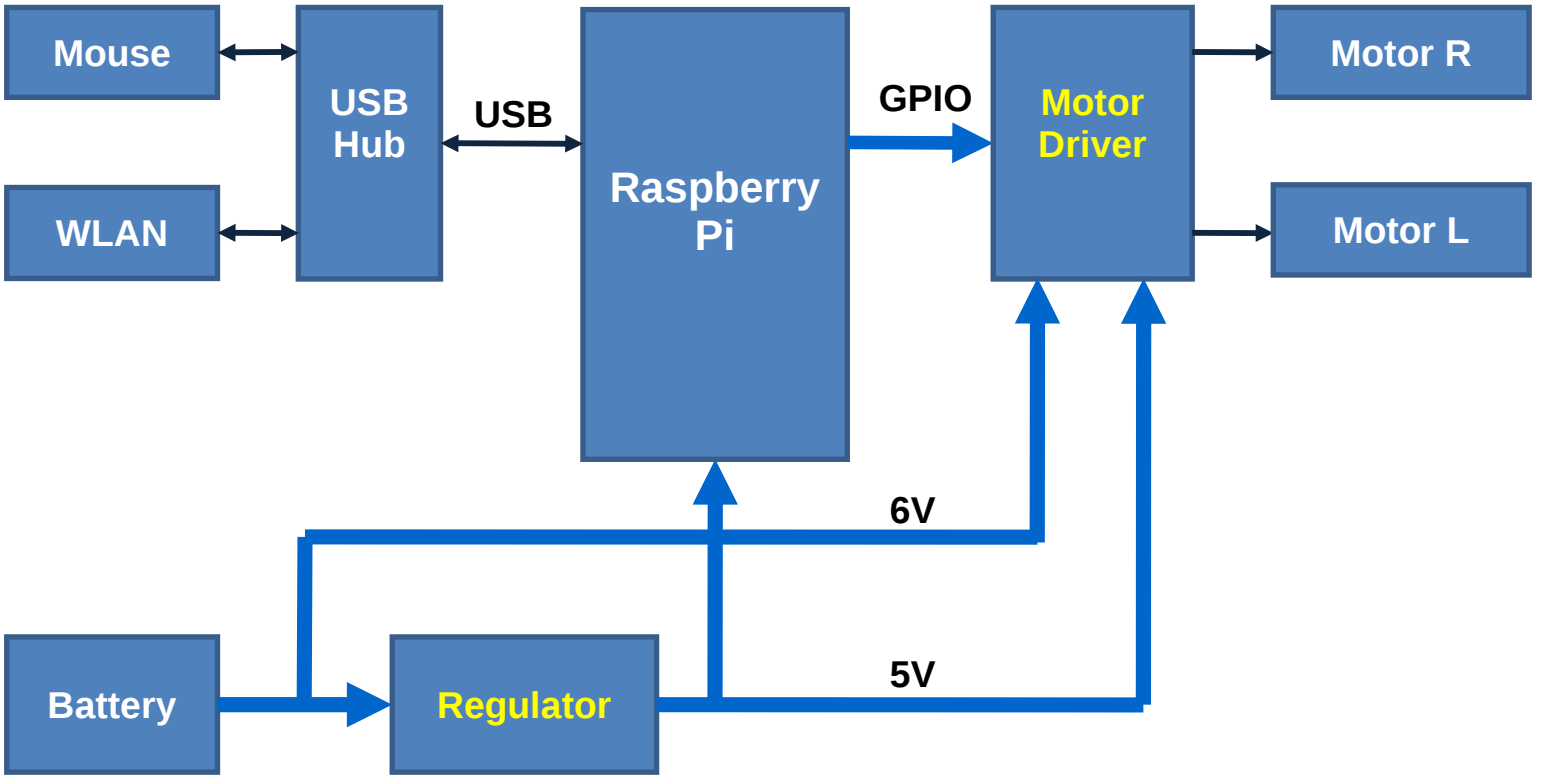
(4) Raspberry Piとモーター制御回路のインターフェース
→Raspberry PiのGPIO直結

(5) バッテリー駆動
→モーター電源は6Vバッテリー直結
→その他は低損失の5Vレギュレーターを使用

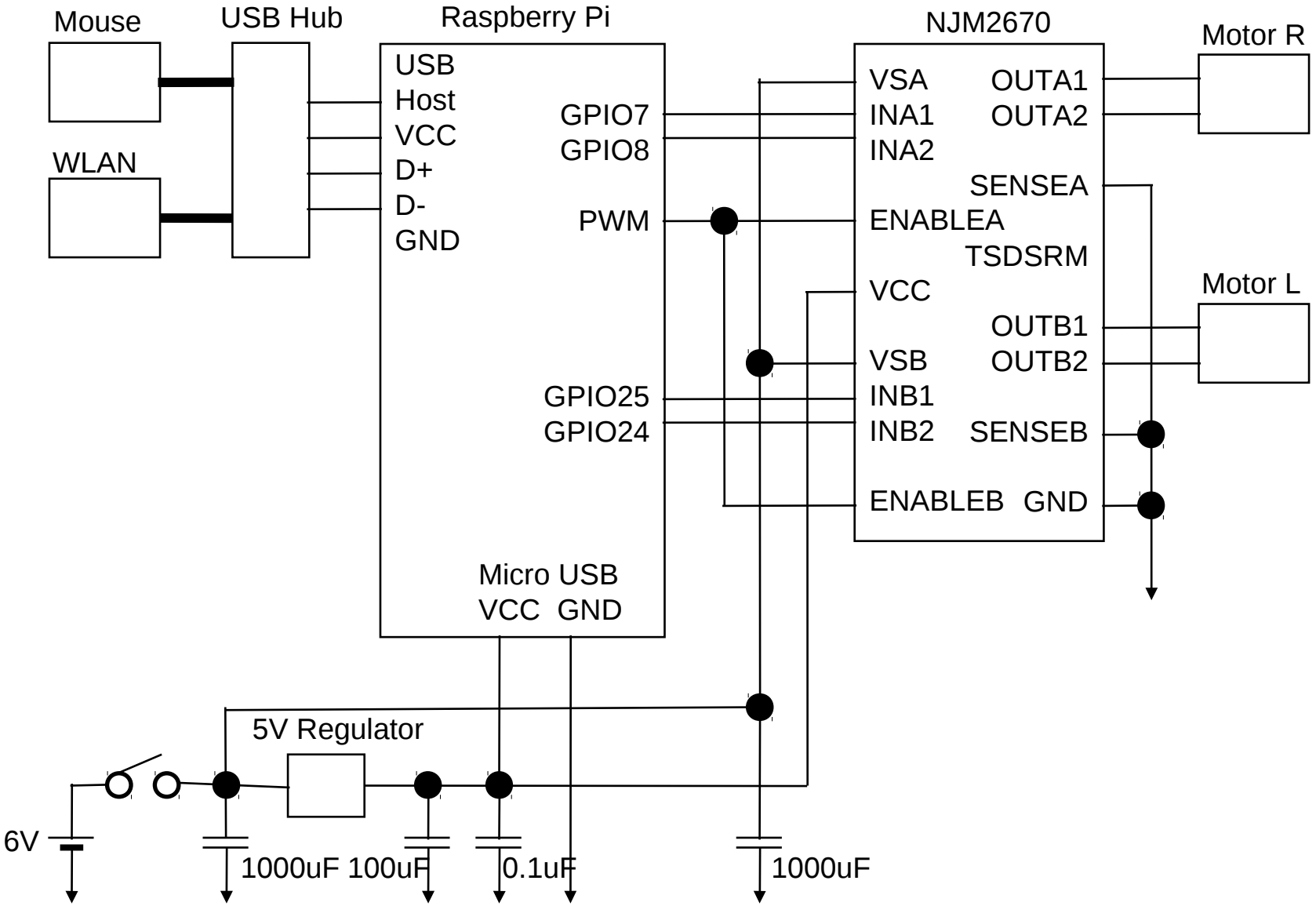
(6) メッセージを指令値とする位置・姿勢制御
→位置計算ノード
→位置制御ノード

ハードウェア構成

TurtleRealのハードウェア構成は以下のようになります。
メインはHブリッジ回路内臓のモータードライバーと電源回路です。



TurtleReal回路図



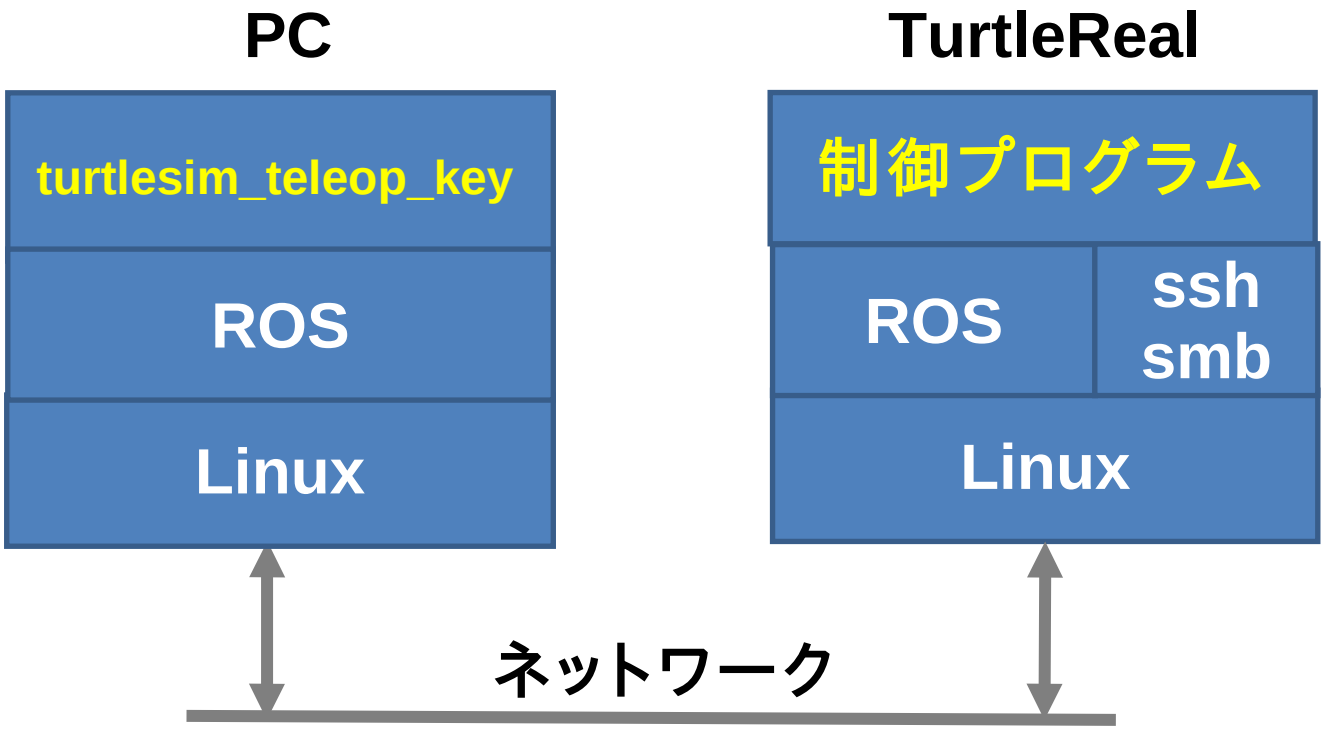
機能の実現手段 (6)

(1) 車輪で位置・姿勢を変える
→2軸車輪による移動
(2) 位置・姿勢検出
→USB光学マウス
(3) 車輪駆動用モーター制御回路
→Hブリッジ回路でPWM制御
(4) Raspberry Piとモーター制御回路のインターフェース
→Raspberry PiのGPIO直結
(5) バッテリー駆動
→モーター電源は6Vバッテリー直結
→その他は低損失の5Vレギュレーターを使用

(6) メッセージを指令値とする位置・姿勢制御
→位置計算ノード
→位置制御ノード

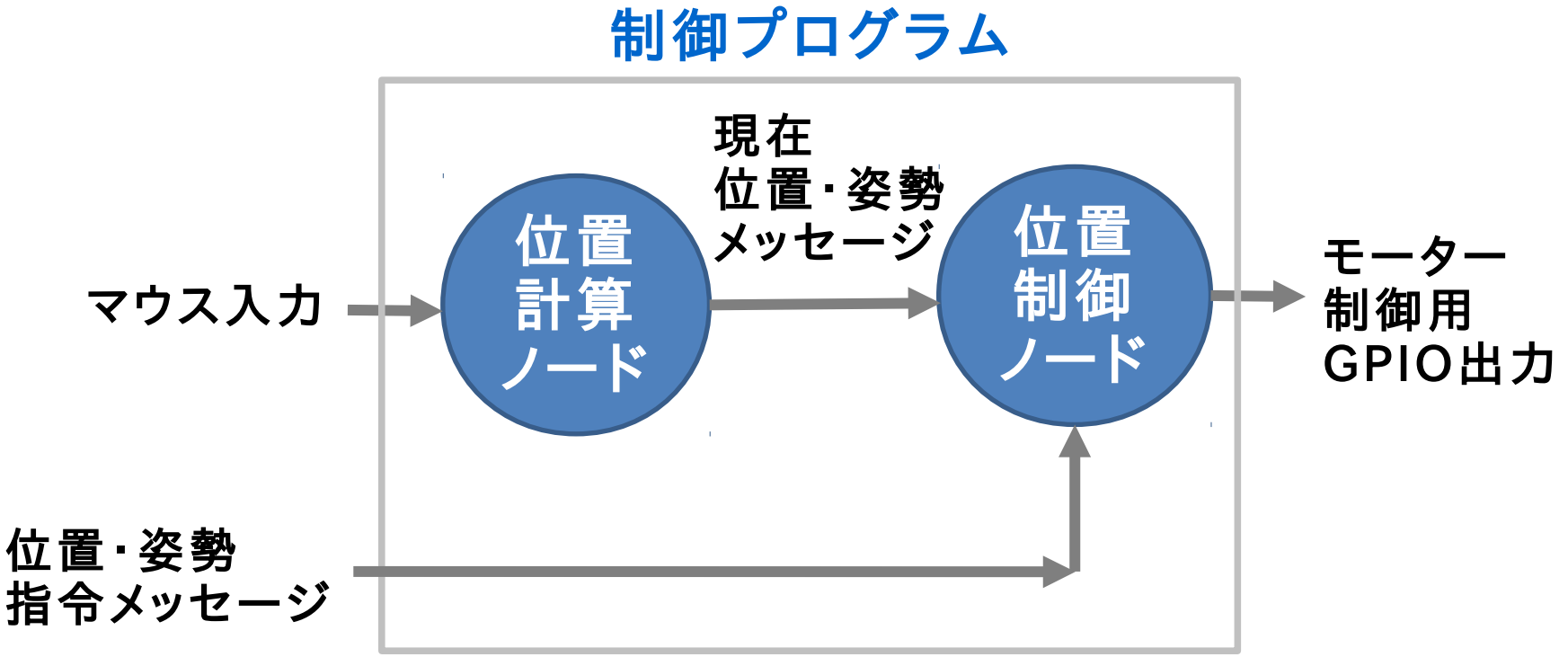
全体ソフトウェア構成

全体のソフトウェア構成は下図のようになります。
PCの「turtlesim_teleop_key」とTurtleRealの
「制御プログラム」がメッセージ通信を行います。



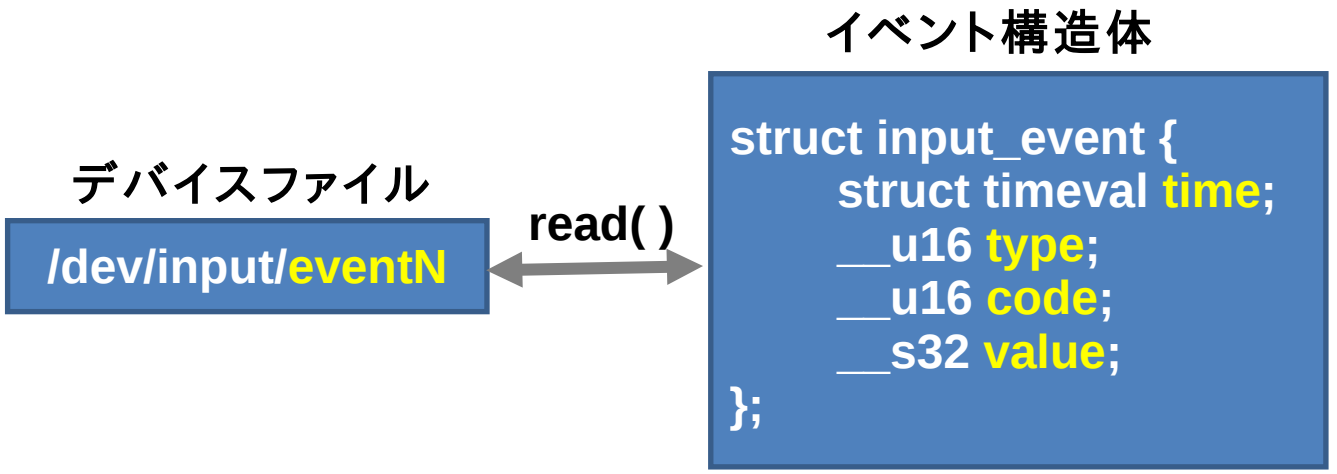
制御プログラム構成

TurtleRealの制御プログラムの構成は下図のようになります。



位置計算ノードのマウス入力

マウスの移動量はデバイスファイル /dev/input/eventN(N>=0)から読み取れます。



time = 入力のあった時間
type = EV_REL
code = REL_X / REL_Y
　REL_X: マウスX軸方向
　REL_Y: マウスY軸方向
value = マウスの相対移動動量

位置制御ノードのGPIO入出力

Raspberry Piでは仮想ディレクトリ/sys/class/gpioを通して
GPIOにアクセス出来ます。

(1) ポートオープン

ポート番号 → Write( ) → /sys/class/gpio/export

(2) 入出力モード設定

入出力方向 ("in" / "out") → Write( ) → /sys/class/gpio/gpio7/direction

(3) ポート出力

出力値 → Write( ) → /sys/class/gpio/gpio7/value

(4) ポート入力

入力値 ← Read( ) ← /sys/class/gpio/gpio7/value

位置制御ノードの位置制御（1）

■モーター軸がウォームギアなので**バックドライバビリティがありません。**（出力軸を回してもモーターは回転しない。）

出典Wikipedia「ウォームギア」より

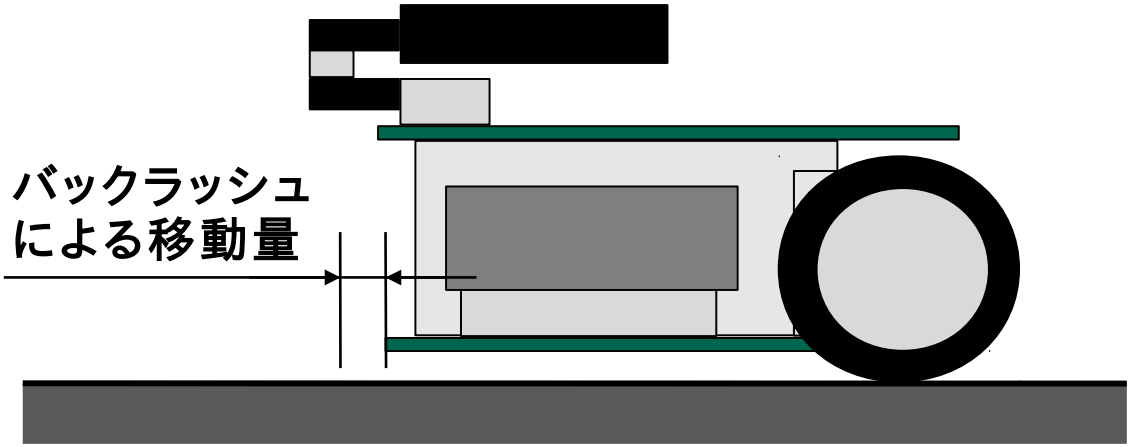
■ローターイナーシャが小さいのでモータドライバーの回生ブレーキにより**モーター自体は急速に減速**します。

■停止指令が出てからモーターが5回転で停止と仮定すると減速比661:1とタイヤ直径58mmから停止距離は下式のようになります。

$$Ls = \pi \times 58mm \times 5回転 / 661 = 1.38mm$$

位置制御ノードの位置制御（2）

■ギアボックスのバックラッシュが実測値では±3mmなので
停止精度を6mmより小さくするのはかなり面倒です。



以上から停止精度の目標は±3mmとしました。

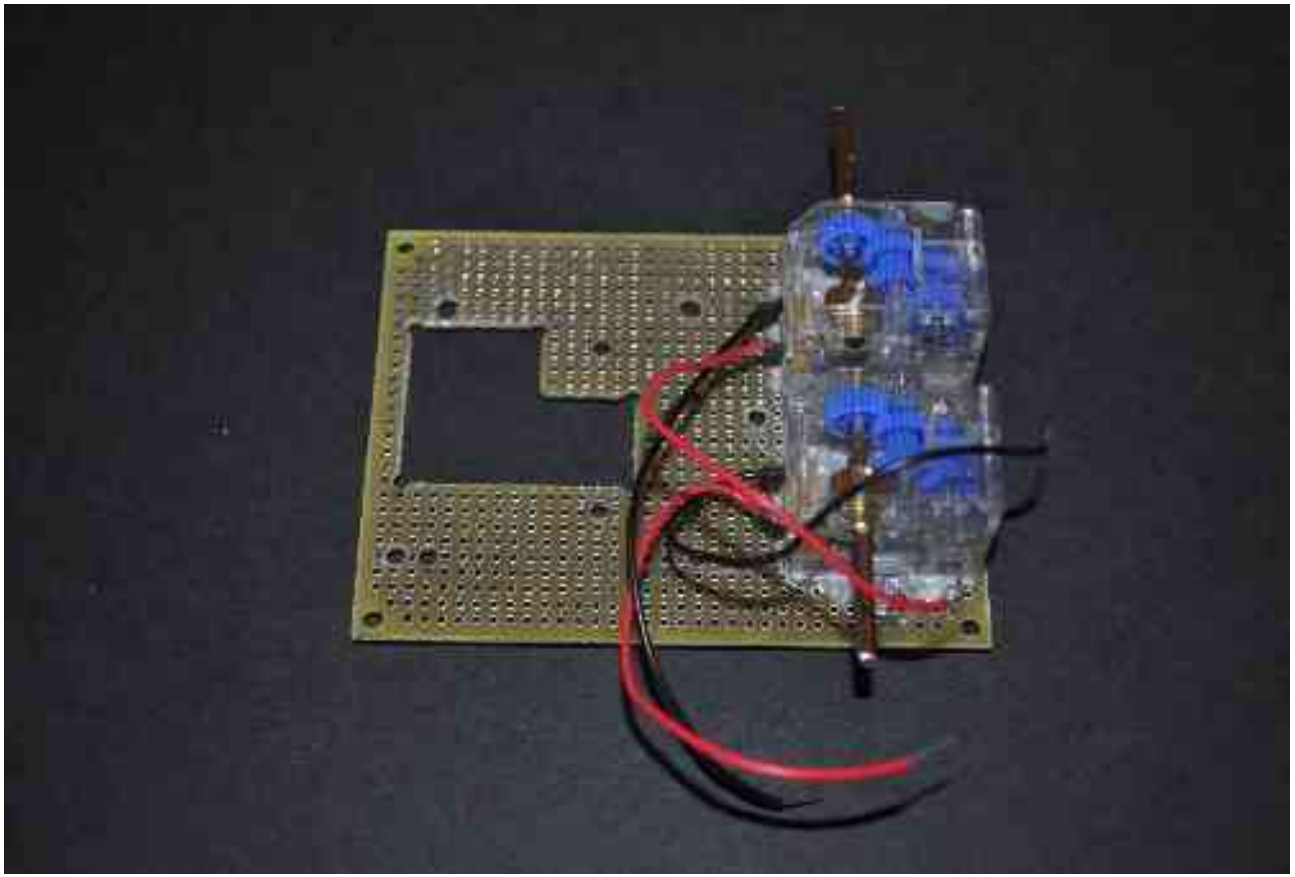
■ロボットの重量も315gと軽いので慣性力の影響も少なく
位置ループ制御は必要なさそうです。（指令位置に近づいたらモーターを停止させるだけで十分な停止精度が出る。）

TurtleSimの製作

TurtleSimの製作

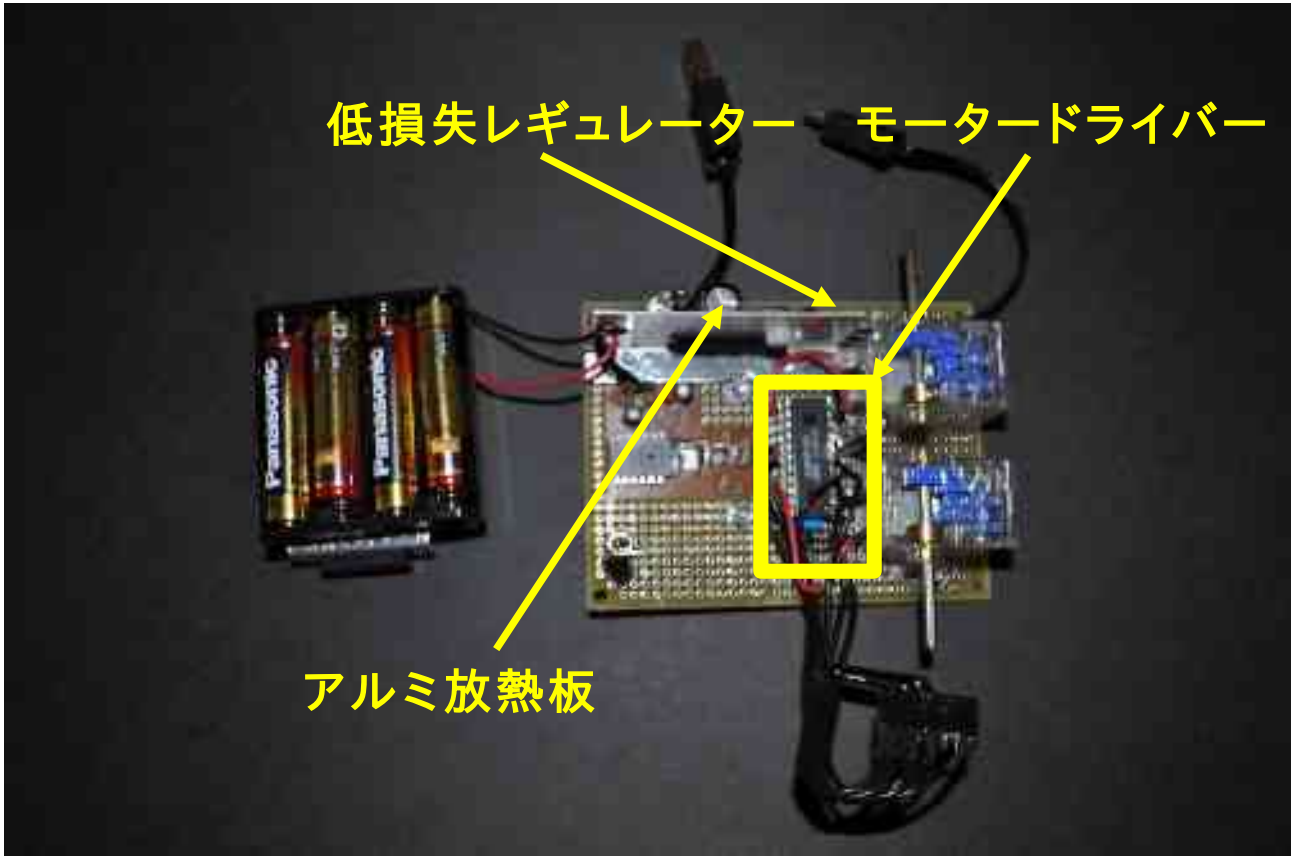
TurtleReal構造（基板兼シャーシーとギアボックス）

ガラスエポキシ基板をシャーシとして小型のギアボックスを2つ付けています。



TurtleReal構造（主要回路）

基板中央にモータードライバーを付け基板右側にアルミの放熱板を立てて低損失レギュレーターを付けています。

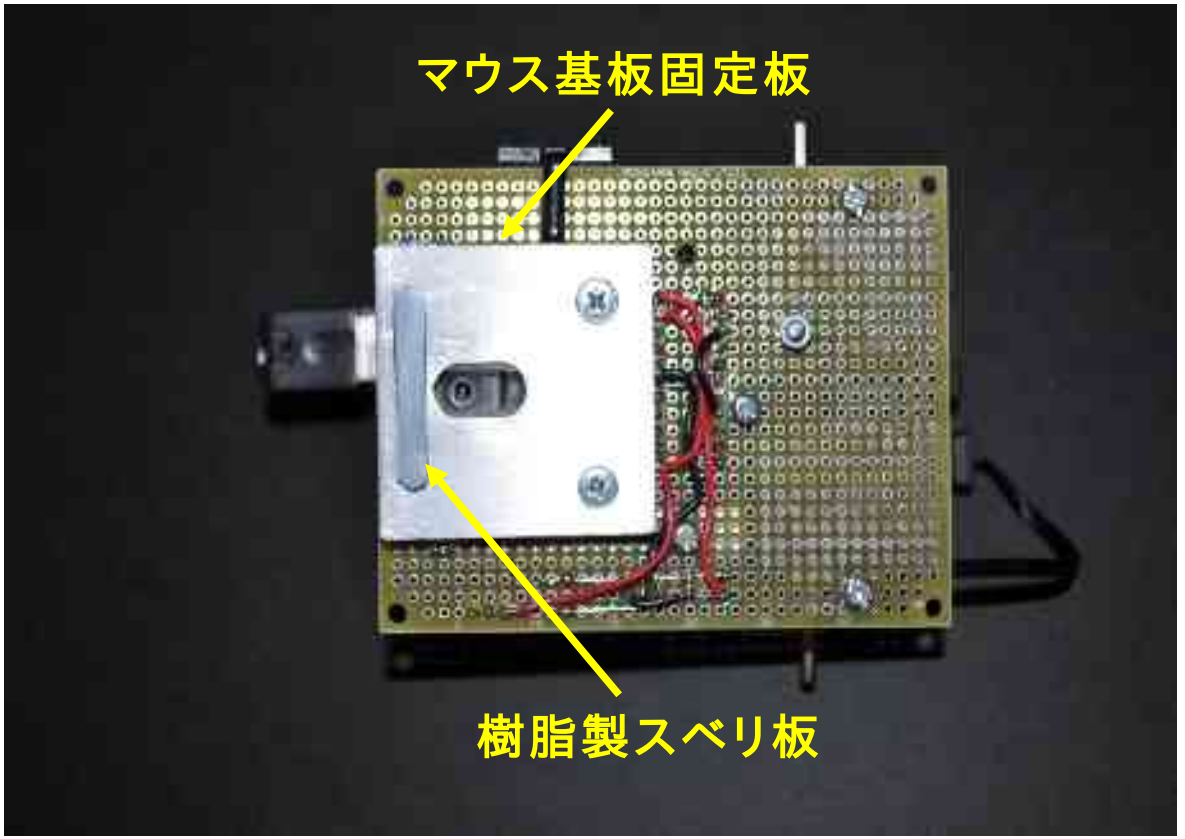


TurtleReal構造（位置検出用マウス基板）

マウスの内部基板です。点線の部分で切り取り、位置検出器のみをロボット底面に固定します。

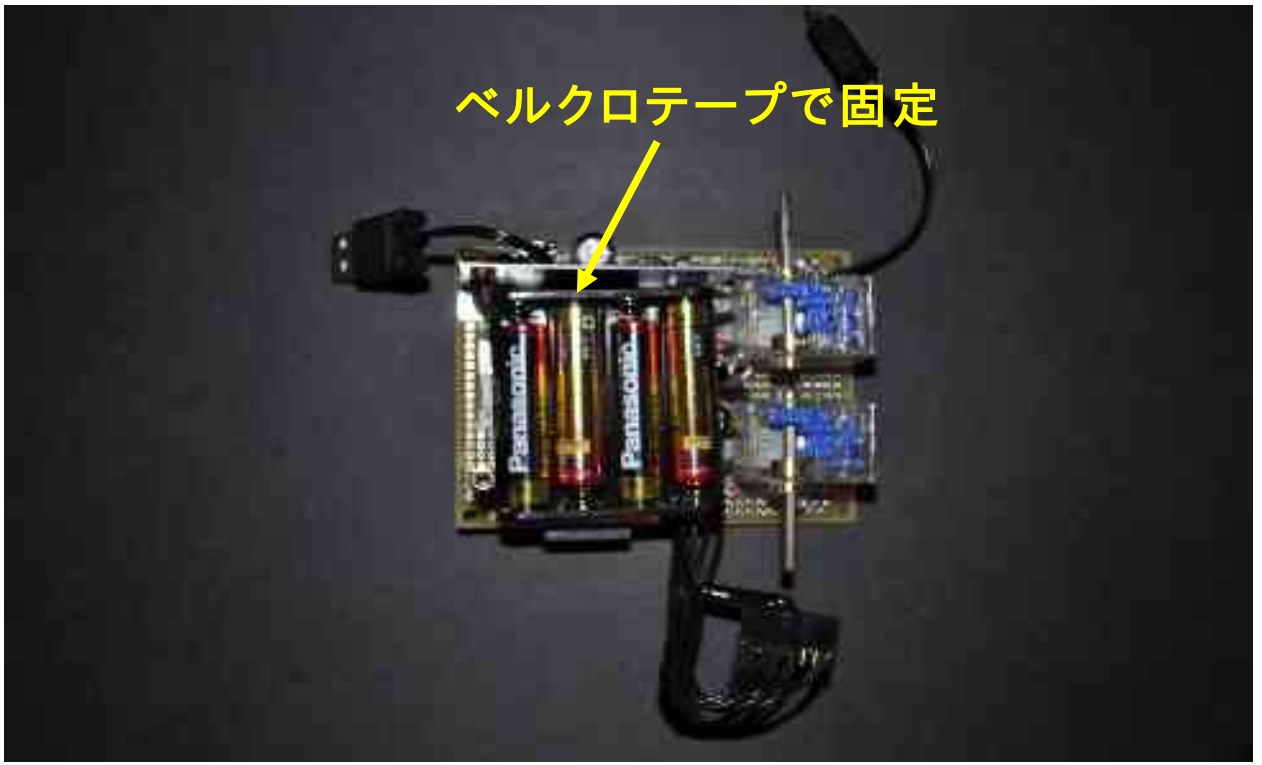
TurtleReal構造（マウス基板取り付け）

位置検出用マウス基板はアルミ板で挟んで底面に固定しています。また、アルミ板にはマウスから剥がした樹脂製スベリ板を付けています。



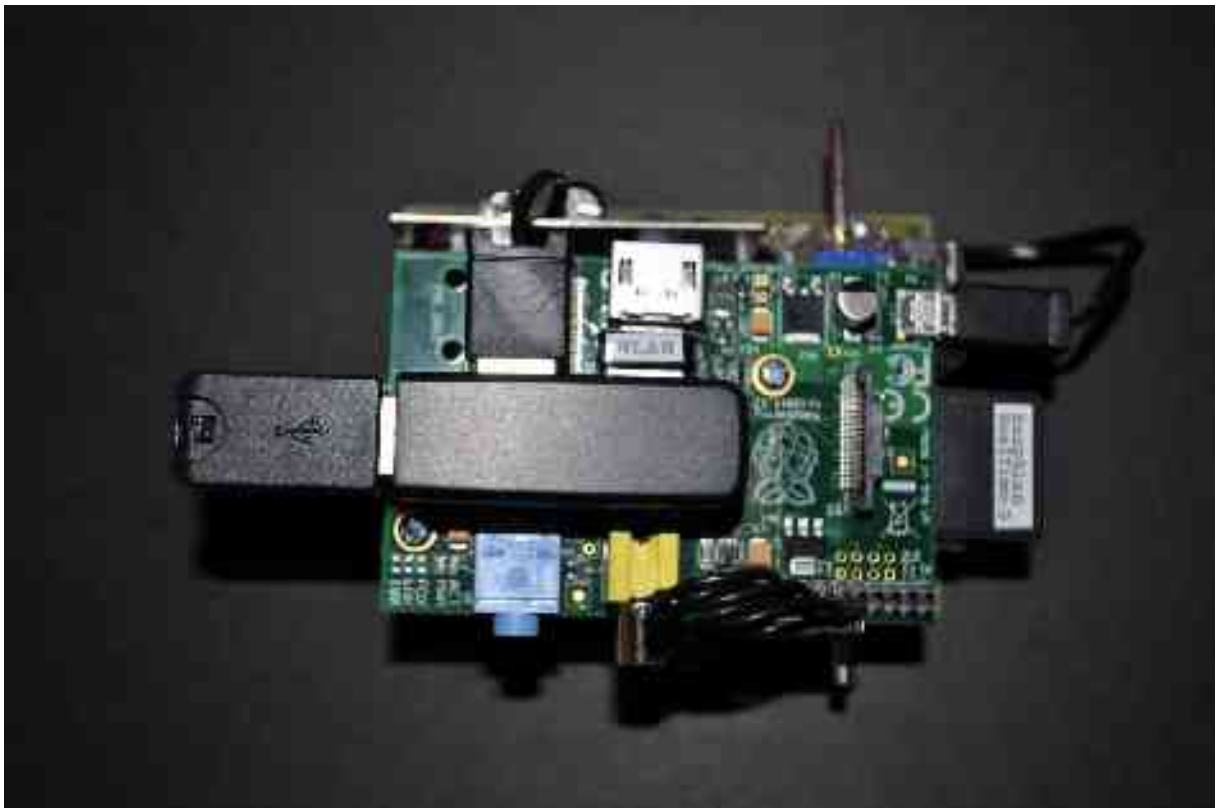
TurtleReal構造（バッテリーケース固定）

バッテリーケースはコの字状に曲げたアルミ板に貼り付け、基板上におきます。バッテリーケースが動かない用にケース片側を放熱板にベルクロテープで固定しています。



TurtleReal構造（Raspberry Pi取り付け）

基板の上にRaspberry Piを金属スペーサーで浮かせて取り付けています。

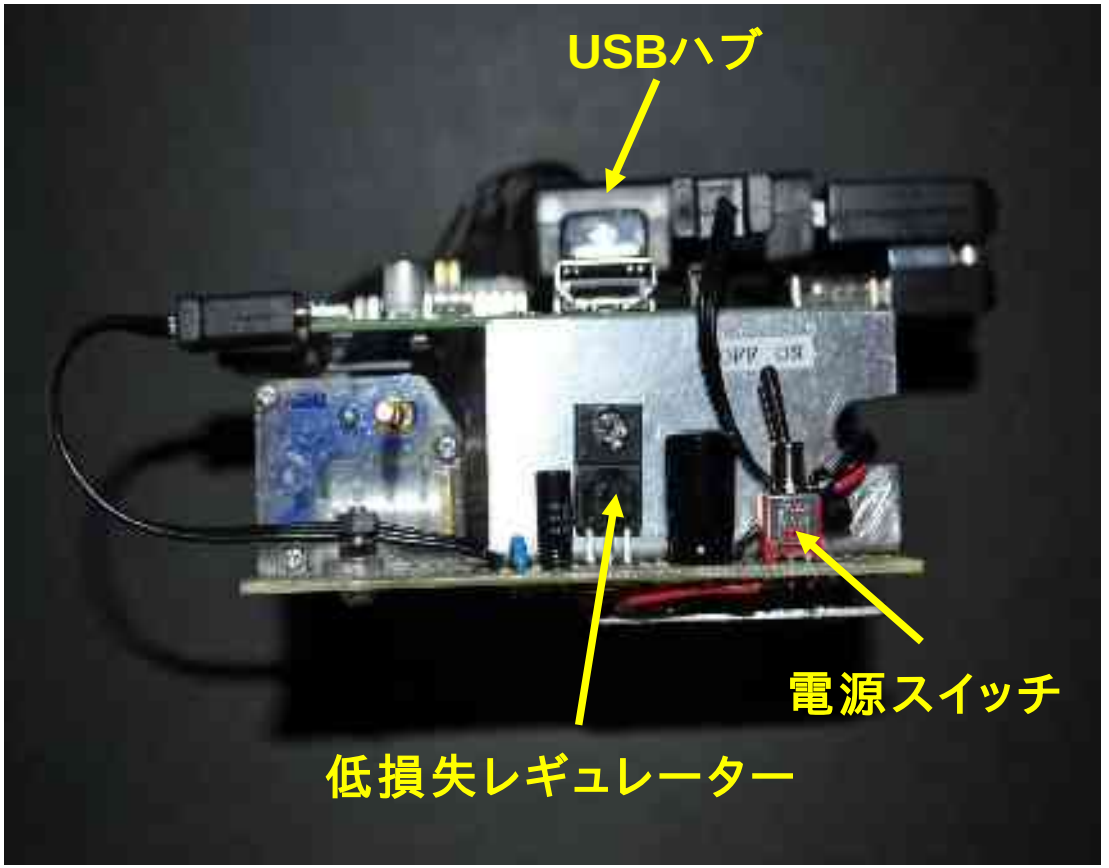


TurtleReal構造（USBハブ）

今回使用したRaspberry PiにはUSBポートが1つしかないので以下のようにしてUSBハブを取り付けます。

TurtleReal構造（本体右側面）

タイヤ以外の全ての部品が付いた状態の右側面です。



TurtleReal構造（本体左側面）

タイヤ以外の全ての部品が付いた状態の左側面です。

TurtleReal構造（完成）

タイヤを取り付けて完成！

主要部品コスト

以下はRaspberry Piと、その周辺を除いたコストです。
小型化にこだわらなければ、あと￥1600は安くなります。

部品名	型番/仕様	販売店	個数	単価
ミニモーター低速ギヤボックス	4速	タミヤ	2	¥928
ナロータイヤセット	58mm径	タミヤ	1	¥518
マウス	USB光学	ダイソー	1	¥300
モータードライバー	NJM2670	秋月電子	1	¥300
基板	両面スルーホール	秋月電子	1	¥200
分割ロングピンソケット	42pin	秋月電子	1	¥80
電池ケース	単4 x 4	秋月電子	1	¥50
5Vレギュレーター	TA4805S	秋月電子	1	¥100
電源スイッチ	125V,5A	秋月電子	1	¥80
電解コンデンサ	1000uF,16V	秋月電子	2	¥20
アルミ板	1.0t 300 x 100mm		1	¥120
			合計金額	¥3644

結論

Raspberry Piと合わせても¥5000程度でROS搭載ロボットが作れます。

TurtleRealデモンストレーション

TurtleRealのデモンストレーション

ご静聴ありがとうございました。

詳しくは「安曇野 ROS」で検索！