コンピュータ理工学特別研究報告書

題目　Raspberry Pi を用いた模型自動車の制御

学生証番号　145307

氏名　牟田　瞭太郎

提出日　平成 2X 年 X 月 XX 日

指導教員　蚊野　浩

京都産業大学
コンピュータ理工学部

要約

組み込み機器の開発に用いる PIC マイコンや Arduino は、パソコンに比べると計算能力が劣り、装置の動作を制御するコントローラとしての利用にとどまっている。一方、パソコンを組み込みシステムに用いることはコストパーフォマンスが悪い。それに対して、Raspberry Pi を用いると、オープンソースソフトウエアなどのパソコンリソースを活用し、組込み装置のプロトタイピングを、パソコンソフトを開発するような手軽さで開発可能になる。本研究では、このような特徴を持つ Raspberry Pi を用い、プログラミング言語 Python によって模型自動車を制御した。

ゲーム機用リモコンを用いて模型自動車を操作するために、これを Raspberry Pi に USB ケーブルで接続する。リモコンは多くのボタンとセンサを持つ。Python プログラムは、それらの信号をリモコンに対応するデバイスファイルから入力する。読み取った信号に従って、模型自動車の動作を制御した。模型自動車は前後移動のために DC モータを用いた。また、方向を変えるステアリングのためにサーボモータを用いた。DC モータを駆動するためのドライバ回路を設計し、そのドライバ回路を Raspberry Pi の GPIO を介して制御する。サーボモータは ServoBlaster とよばれるデーモンプログラムを呼び出すことで制御する。また、カメラモジュールを取り付け、顔検出することにより、顔を検出した場合、模型自動車のモータを止めるプログラムも作成した。

本研究によって、Raspberry Pi を用いることで、パソコンによるシステム開発のような手軽さで組み込み機器を開発することができた。しかもそのコストはパソコンを利用した場合よりも、はるかに小さいものであった。

目次

1章　序論

コンピュータの主要な利用分野の一つに、電気機械装置にコンピュータを組込んで、高度な機能や複雑な制御を実現するものがある。代表的な例は冷蔵庫やエアコンなどの家電機器、自動車・飛行機・ヘリコプタなどの輸送機器、産業用途で利用される製造装置や検査装置、医療で利用されるX線CTやNMR、駅の自動改札や券売機などである。これらの装置では、コンピュータがセンサ・可動部品・表示装置と一体化して動作するため、利用者はコンピュータを使っている意識が少ない。このような機器を「組込み機器」とよび、このように利用されるコンピュータシステムを「組込みシステム」とよぶ。

組込みシステムで利用されるコンピュータ技術は多岐にわたる。家電製品ではパソコンやサーバーで利用される技術と比較して、高い計算能力を必要としないがコストに対する制約が強い。そのため安価なワンチップマイコンと，必要最小限の周辺回路を加えた構成になることが多い。自動車に利用される組込みシステムは価格に対する制約とともに、動作の信頼性が重要になる。X線CTやNMRなどの医用画像装置は高い計算能力が必要である。これらの用途では、高性能パソコンを機器に組込むことも多い。このように、組込みシステムは用途に応じたノウハウが必要で、開発に高い専門性が必要である。

比較的容易に組込み装置を開発することができるコントローラとしてPICマイコンやArduinoが知られている。PICは、1980年代から利用されており、CPU、メモリ（ROM、RAM）、I/Oが収められた1チップマイコンである。C言語でプログラムを開発する。Arduinoは、2005年頃から入手可能になったマイコンボードである。PICと同様の1チップマイコンに周辺回路を加え、14本のデジタルI/Oピン、12本のアナログI/Oピンを備えている。Arduinoはオープンプラットフォームであるため、多くのソフトを無償で利用することができる。PICはプロトタイピング（試作）から最終製品まで利用され、Arduinoはプロトタイプでの利用が多い。PIC、Arduinoともに、パソコンに比べると計算能力が著しく劣るため、装置の動作を制御するコントローラとしての利用にとどまっており、また、パソコンソフトを動作させることも難しい。

Raspberry Piは、2012年から入手可能になったマイコンボードである。外観はArduinoと似ている。外部装置と接続するためのI/O端子を備えており、コントローラとして利用することができる。それだけでなく、ディスプレイ・キーボード・マウスを接続すると、完全なLinuxパソコンとして利用することが

できる。そのため、オープンソースソフトウエアなどのパソコンリソースを活用することができる。これらのことから、Raspberry Pi を用いると組込み装置のプロトタイピングを、パソコンソフトを開発するような手軽さで行うことができる。本研究では、このような特徴を持つ Raspberry Pi を用いて模型自動車をリモコンで制御することを行った。

本論文は以下のように構成される。2 章では Raspberry Pi の特長と Raspberry Pi を用いた外部機器の制御について述べる。3 章では Raspberry Pi を用いた模型自動車のリモコンによる制御について述べる。4 章では本研究で行ったリモコンによる制御、カメラ信号を用いた制御、ステアリングの制御に関して述べる。最後に 5 章で結論を述べる。

2 章　Raspberry Pi と機器の制御

この章では、Raspberry Pi の特徴、Raspberry Pi を用いて機器を制御するための基本的な方法、および、本研究で用いるプログラミング言語である Python について説明する。

2.1 Raspberry Pi の特徴

Raspberry Pi は、カードサイズの CPU ボードである。Linux 系 OS が動作し、ディスプレイ、キーボード、マウス、ネットワークを接続することで一台のコンピュータとして利用することができる。また、ボード上に幾つかのコネクタを備えており、組込み機器のプロセッサボードとして利用することもできる。図 1 に Raspberry Pi に搭載されている主な部品の名称と機能を示す。SD カードはシステムプログラムとユーザプログラム、データを格納する外部メモリとして利用する。CPU は ARM11 である。ボード中央にあるチップは、CPU を始めとする多くの機能を一つのチップに集積した SoC（System on Chip）である。ディスプレイ、キーボード、マウス、イーサーネットは、それぞれ HDMI コネクタ、USB 端子、イーサーネット端子に接続する。電源はマイクロ USB の電源コネクタを介して供給する。電源スイッチは無く、コネクタに電源プラグを接続することでパワーが供給される。電源を切る場合にはあらかじめシステムをシャットダウンし、動作が止まったことを確認して電源プラグを抜く。また、ピンヘッダやコネクタを介して外部の機器・装置・センサ・メカを接続できる。

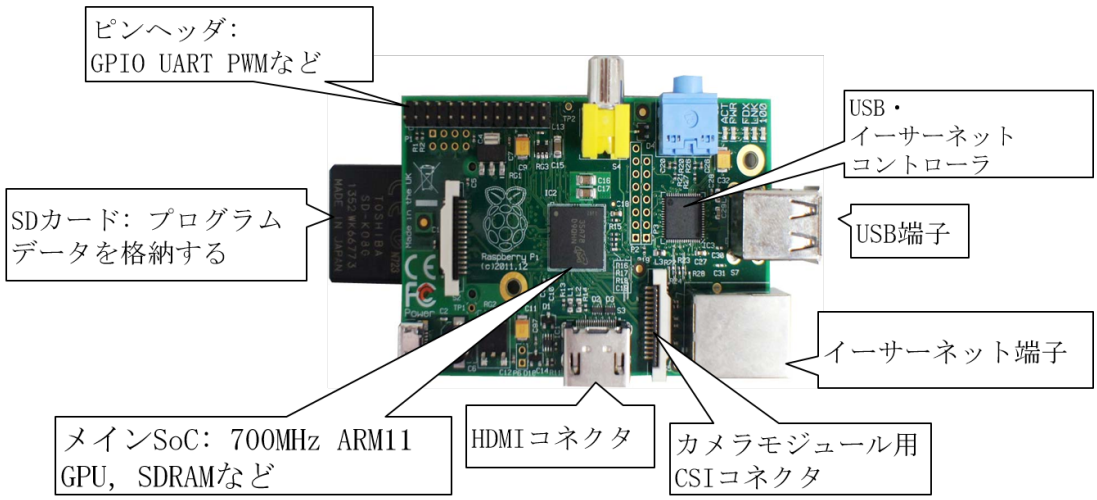
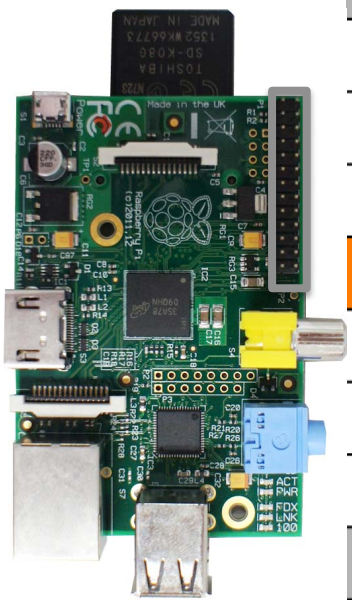


図 1　Raspberry Pi　TypeB ボードの外観

2.2 Raspberry Pi による機器の制御

Raspberry Pi が普通の Linux パソコンと最も異なっている点は、汎用入出力（GPIO、General Purpose I/O）とよばれる電子回路と直接接続できる端子を持つことである。GPIO は，Raspberry Pi 基板上の 40 本のピンヘッダからアクセスする。40 本のピンの割り当てを図 2 に示す。GPIO 端子の多くは汎用入出力以外の機能も持っている。また、ピンヘッダから 3.3V、5V の電圧を取り出すことができる。



+3.3V	P1	P2	+5V
GPIO 2	P3	P4	+5V
GPIO 3	P5	P6	GND
GPIO 4	P7	P8	GPIO 14
GND	P9	P10	GPIO 15
GPIO 17	P11	P12	GPIO 18
GPIO 27	P13	P14	GND
GPIO 22	P15	P16	GPIO 23
+3.3V	P17	P18	GPIO 24
GPIO 10	P19	P20	GND
GPIO 9	P21	P22	GPIO 25
GPIO 11	P23	P24	GPIO 8
GND	P25	P26	GPIO 7

図 2　Raspberry Pi のピンヘッダとピンアサイン

2.3 Python の特徴

Python は様々な分野のアプリケーションで使われる、パワフルで動的なプログラミング言語である。Python には次のような特徴がある。直感的なオブジェクト指向、手続きコードによる自然な表現、パッケージの階層化をサポートした完全なモジュール化をサポートする、例外ベースのエラーハンドリング、高レベルな動的データ型などである。[1]

Raspberry Pi のプログラミング言語として C/C++・Java を使うこともできるが、この実験では Python を使う。Python は初心者向けの言語として開発されたものであり、構文がシンプルで修得が容易である。現在では、非常に多くの拡張機能が開発されており、さまざまな用途に利用することができる。また、インタプリタで実行されるため、C/C++や Java のようにコンパイルやビルドを行って実行ファイル（機械語のプログラム）を作る必要がない。作成したプログラムは Python のインタプリタによって直接実行される。一般にインタプリタは素早くプログラムを作成するのに向いているが、実行速度は遅くなる。

3 章　Raspberry Pi とリモコン、模型自動車の接続

この章では、Raspberry Pi とリモコン、カメラモジュールとの接続、及び、本研究で用いた模型自動車の構成要素、駆動回路について説明する。

3.1 リモコンと Raspberry Pi の接続

本研究では、ゲーム機 PS3 用の市販リモコンを用いて模型自動車を制御した。このリモコンは USB ケーブルによる有線接続と WiFi による無線接続が可能である。今回は Raspberry Pi の USB 端子に有線接続した。

図 3 にリモコンの外観を示す。これらのボタンの ON/OFF と、ON の場合のボタンへの圧力（押し込み量）、およびジョイスティックの位置を検出することができる。また、加速度センサなどを備えており、それらのセンサ信号を検出することができる。今回の実験で利用したボタンの信号を Python プログラムに入力する方法については 4 章で説明する。

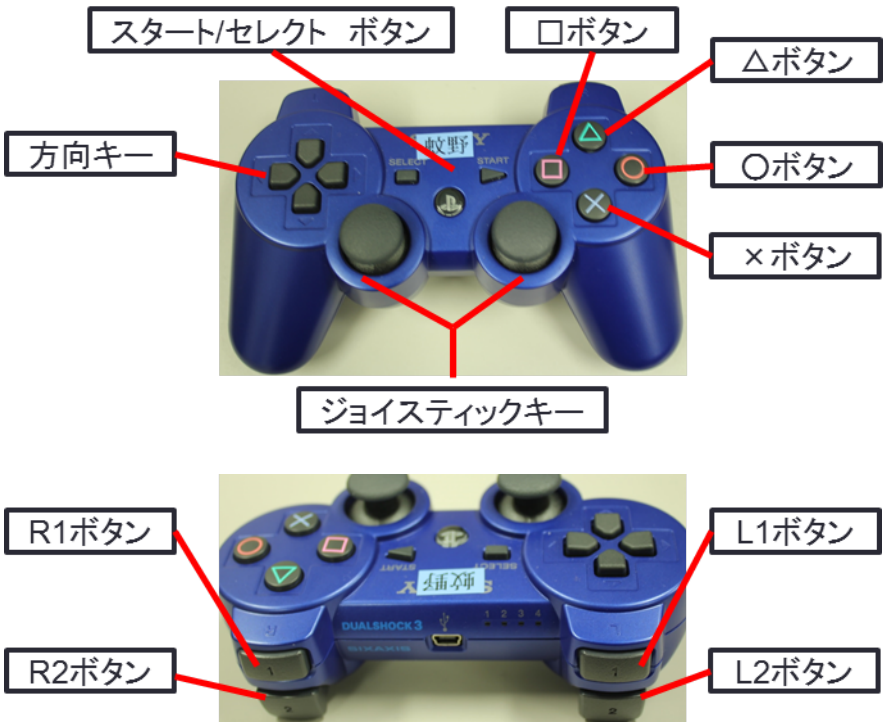


図 3 PS3 コントローラのボタンの種類

3.2 Raspberry Pi とカメラモジュールの接続

Rasspberry Pi 専用のカメラモジュールは図 4 のもので、撮像素子と呼ばれる半導体とレンズ・周辺部品をプリント基板に一体化したものである。撮像素子には、5M ピクセルの CMOS センサが使われる。カメラモジュールは、15 ピンのフラットケーブルで Raspberry Pi ボードの CSI(Camera Serial Interface)

コネクタに接続する。CSI コネクタのカバー(白い部分)を引き上げ，フラットケーブルを金属端子が図の手前側に来るように挿入する。ケーブルを奥まで差し込んだ状態で、カバーを押し込む。カメラモジュールの画像・映像信号を Python プログラムに入力する方法については、4 章で説明する。

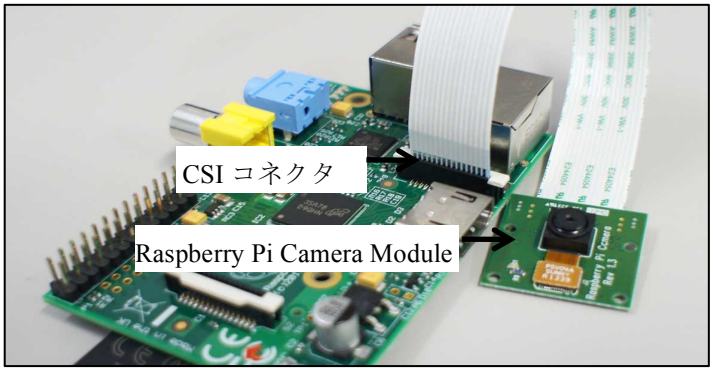


図 4　カメラモジュールと Raspberry Pi への接続

3.3　模型自動車の構成要素

この実験で使う模型自動車の構成要素は、Raspberry Pi、PS3 コントローラ、車輪 4 個を装着した台車、後輪を駆動する 2 個の DC モータとモータの動きを後輪に伝達するギアボックス、DC モータを駆動する回路、前輪のステアリングを制御するサーボモータと電池ボックスである。それらの接続関係を図 5 に、それぞれの外観を図 6 に示す。これらの部品は、TAMIYA 社の楽しい工作シリーズ No.97「ツインモータギヤーボックス」、同 No.98「ユニバーサルプレートセット」、同 101「トラックタイヤセット（36mm 径）・4 本セット（ホイールつき）」を用いた。

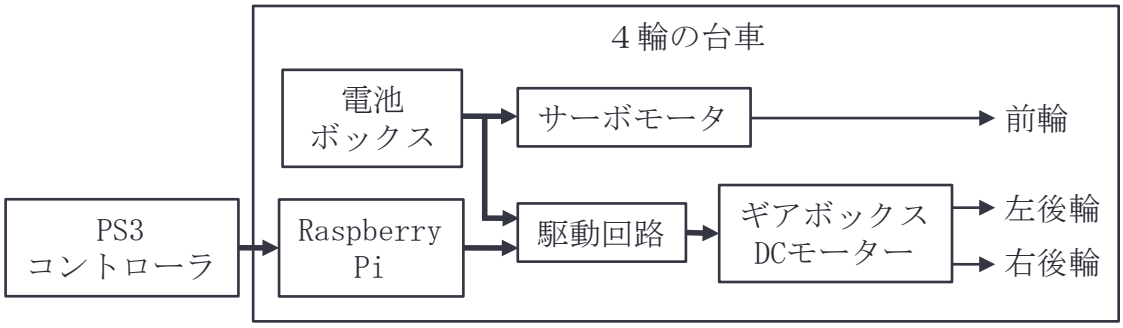


図 5 模型自動車を構成する要素の接続関係

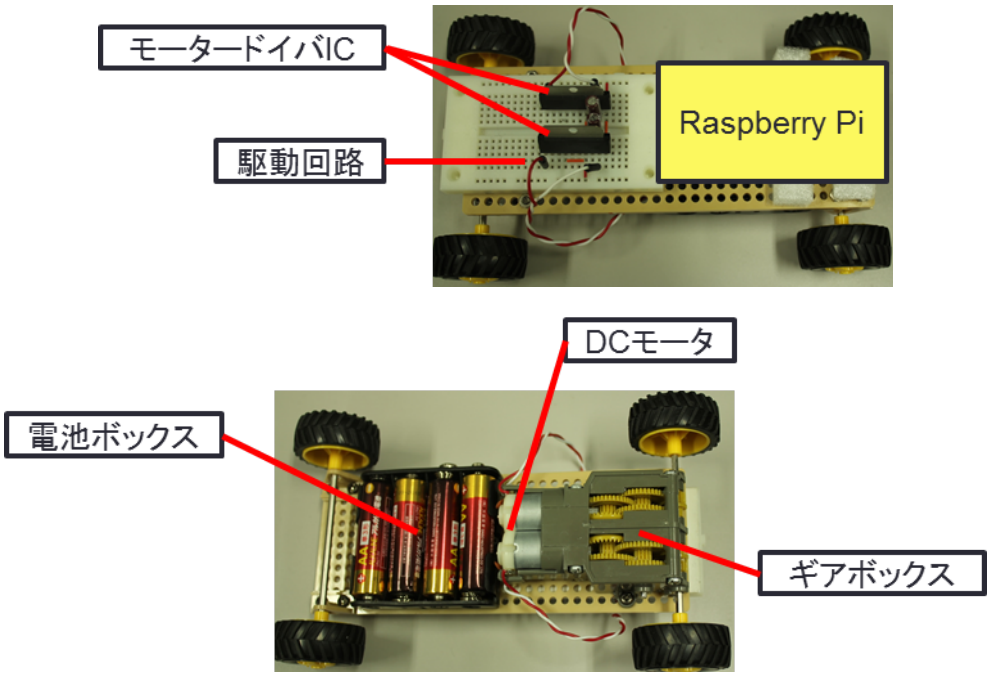


図 6 それぞれの構成要素ごとの外観写真

3.4　模型自動車の駆動回路

Raspberry Pi の GPIO ピンから出力される電流量では，後輪を回転させる DC モータを十分なトルクで駆動することができない。そこで、モータを駆動するための電池を外付けし、モータドライバ IC を使う。GPIO の出力はモータドライバ IC を制御するために用いる。図 7 に駆動回路の回路図を示す。この回路で、モータドライバ IC の IN1・IN2 端子に加える信号とモータに流れる電流の関係を図 8 に示す。サーボモータへの制御信号は、GPIO 端子を直結した。サーボモータの制御については、4 章で説明する。

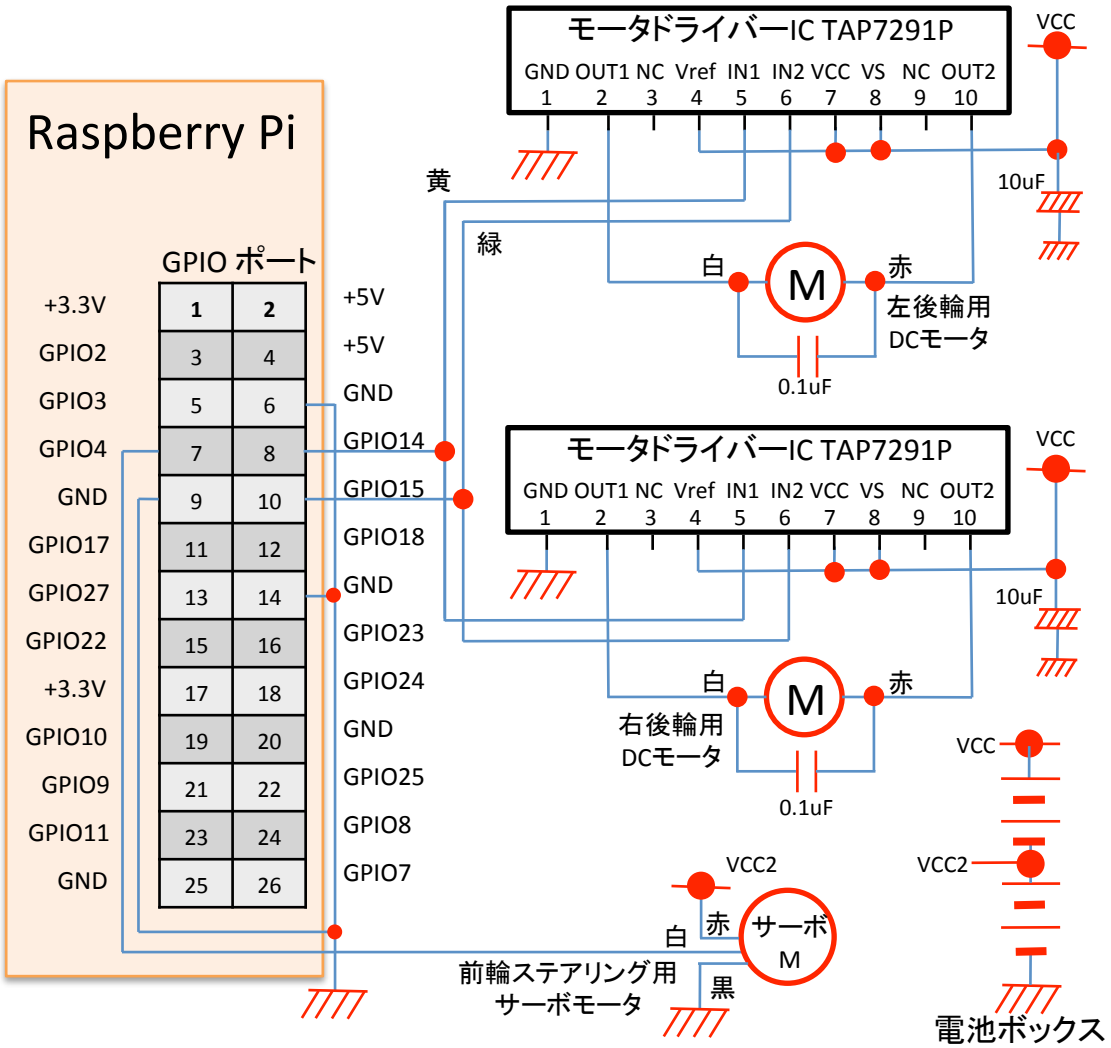


図 7 模型自動車の駆動回路

表 1 モータドライバ IC の各端子の説明と
IN1・IN2 端子への入力信号と機能の関係

ピン	TA7291P	接続
VCC	7	ドライバーICの電源
VS	8	モーター用電源
Vref	4	モーターの電源コントロール用
GND	1	
IN 1	5	Raspberry Piにつなぐ信号1
IN 2	6	Raspberry Piにつなぐ信号2
OUT 1	2	モーターにつなぐ
OUT 2	10	モーターにつなぐ

IN1	IN2	OUT1	OUT2	機能
0	0	∞	∞	ストップ
1	0	H	L	CW
0	1	L	H	CCW
1	1	L	L	ブレーキ

4 章　実験結果と考察

この章では、今回の実験結果と考察について述べる。

4.1　リモコンによる制御

Rasberry Piから PS3 用リモコンの信号を読取る Python プログラムを開発した。サンプルプログラムの一例を付録 1 に示す。これは、ゲーム用の Python モジュールである pygame を使い、リモコンが押された時に、その動作をグラフィカルに表示するプログラムである。このプログラムでは、初めに sys モジュールを呼び出す。sys モジュールは、インタプリタで使用・管理している変数や、インタプリタの動作に深く関連する関数を定義している。また、画面表示のための初期設定を行っている。

リモコンからの信号はデバイスファイル/dev/input/js0/に 8 バイト単位の信号として送られてくる。その信号を読み取り、内容によりボタンの種類を判別する。このプログラムでは、ボタンの ON/OFF だけでなく、ボタンを押した圧力を読み取っている。このプログラムにおけるボタンの種類とデータの内容関係を簡単にまとめたものを表 2 に示す。なお、ボタンの種類とそれに対応する data[7]の値は、data[6]の値によって変化するので注意が必要である。この不自然な現象は、リモコン動作の解析が不十分なことが原因であると思われる。

表 2 ボタンと data[7]との対応関係

コントローラとボタンとの関係	
ボタンの種類	**data[6]=' 02'の場合 data[7]の値**
□ボタン	' 13 '
△ボタン	' 10 '
○ボタン	' 11 '
×ボタン	' 12 '
R1ボタン	' 0F '
R2ボタン	' 0D '
L1ボタン	' 0E '
L2ボタン	' 0C '

4.2 カメラ信号を用いた制御

カメラモジュールを Raspberry Pi に接続し、撮影した映像を用いて模型自動車を制御する Python プログラムを開発した。そのサンプルコードの一例を付録 2 に示す。カメラモジュールを Python から利用するために io モジュールと picamera モジュールを用いる。io モジュールはカメラから連続的に入力される映像を Python にストリームとして取り込むためのモジュールである。picamera モジュールは今回利用したカメラ用に開発されたものである。import picamera とすることでカメラモジュールが利用可能になる。

カメラモジュールから取り込んだ画像情報を、Python 用の OpenCV を用いて処理することができる。サンプルプログラムでは、カメラで撮影した画像から図 8 のように顔を検出し、顔を検出した場合に、モータを停止させている。

OpenCV を用いた顔検出は、160×120 画素の画像から顔を検出する処理に 1 秒間に 3 枚程度しか処理できない。実験の計画段階では、模型自動車を移動さ

せながらカメラから入力した画像・映像信号を処理することで環境認識することを予定していた。しかしながら、十分の高速な処理を行える目処が立たないため、この計画は中断することにした。

図 8 顔検出の結果

4.3 ステアリングの制御

前輪をステアリングさせるための機構は、TAMIYA 社の楽しい工作シリーズ No.112「バギー工作基本セット」のアッカーマンタイプステアリング機構を用いた。これは、図 9 のもので、回転自在に取り付けられた前輪をシャフトで連結したものである。シャフトを左右に移動させると前輪の方向が変化する。シャフトを左右に移動させるために、L 字状の金具をシャフトに半田付けし、この金具をサーボモータで左右に回転させる。

サーボモータは、回転速度や位置を容易に制御することができるモータである。今回使用したものは、GWS サーボ／STD／F であり、適当なパルス幅の繰り返し波形を加えることで、位置を制御できる。

具体的なサーボモータの制御には、servoblaster というソフトウエアを用いた。これをあらかじめ起動しておくと、デバイスファイル/dev/servoblaster に GPIO のボート番号とパルス幅を指定する数値を送ることで、その GPIO 端子から必要なパルス幅の繰り返し波形が生成される。これを利用してサーボモータを制御するソースコードを付録 3 に記載する。

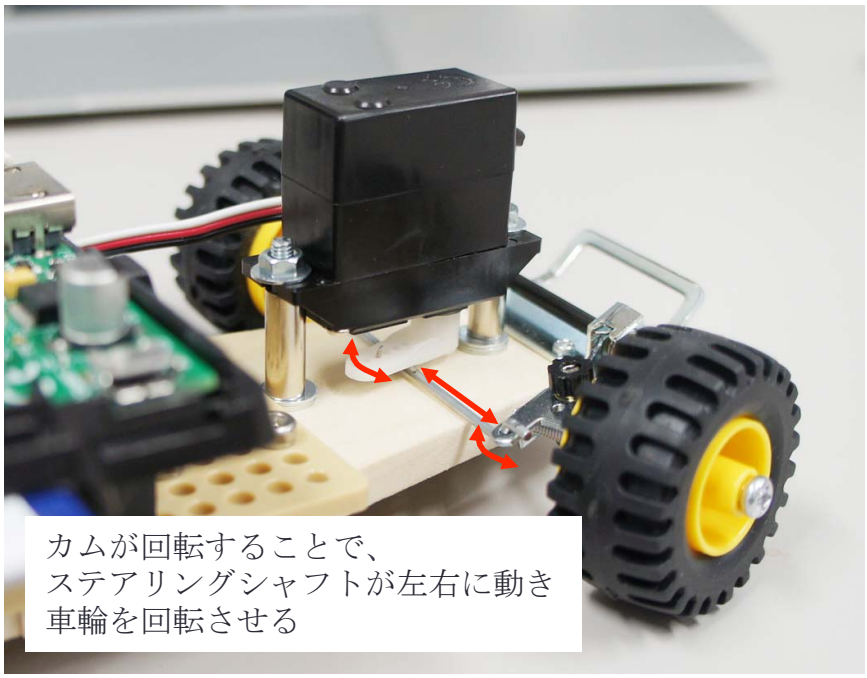


図 9 前輪のステアリング機構とそれを駆動するサーボモータ

4.4 リモコンを用いた模型自動車の制御

ここまでに開発した要素技術をシステム化し、リモコンで模型自動車を自在に操作できようにした。図 10 に完成した装置の外観を示す。このシステムを動作させるサンプルプログラムを付録 4 に示す。

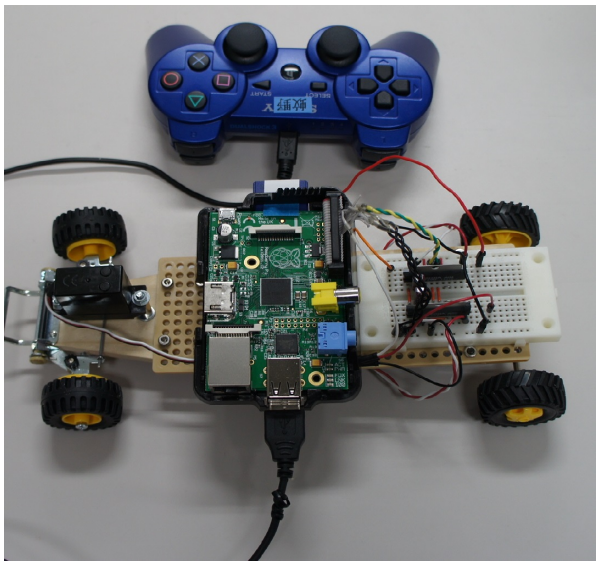


図 10 完成した模型自動車の外観

5 章　結論

リモコンとの接続により、Raspberry Pi にどのボタンが押されたか認識し、処理をするということができた。カメラモジュールの制御では、Python を用いて映像をストリーム処理することができた。また、OpenCV を用いることにより顔検出を行うことが出来た。模型自動車の駆動回路の設計により模型自動車を正しく動かすことが出来た。ステアリングの制御では、サーボモータを使用することで、左右の運転を可能にした。

今回の実験では、リモコンの制御とカメラの制御を同時に行い、自在に動かしながら顔を検出し、顔が検出された場合止まるという同時並行の処理ができなかった。よってリモコンを高速処理しつつ、カメラの処理を行うという Python での並行処理を試すことが課題としてあげられる。また、ステアリングの制御で用いたサーボモータのプログラムは、プログラム自体に問題はないと思うが、実行時にタイムラグが生じることがあり、その場合にはスムーズな動きが出来ない。その原因は、現在のところ不明である。

今回の研究で、Raspberry Pi を用いることによってより、簡単に組み込みシステムにおけるマイコンとしての役割を果たすことが出来ることが分かった。今後将来様々な用途にこの Raspberry Pi を用いることが可能だと考えている。

謝辞

本研究を進めるにあたり、担当教授の蚊野浩先生に感謝致します。丁寧かつ熱心なご指導本当にありがとうございました。

参考文献

[1] 日本 Python ユーザ会　　http://www.python.jp/

付録 1: リモコンの動作を確認するためのサンプルコード

```
# -*- coding: utf-8 -*-
import sys
import pygame

#DualShock3 からの入力コードを/dev/input/js0 から読み取る
file = open('/dev/input/js0','r')
data = []  #8 バイト単位でデータを読み取るバッファ

#表示に用いる定数
black = 0,0,0
gray = 128,128,128
white = 255,255,255
line_width = 3

#○ボタンの定義
class Circle(pygame.sprite.Sprite):
        def __init__(self,x,y):
                pygame.sprite.Sprite.__init__(self)
                self.image = pygame.Surface([30,30])
                self.image.fill(white)
                pygame.draw.circle(self.image,gray,[15,15],15,0)
                pygame.draw.circle(self.image,black,[15,15],15,line_width)
                self.rect = self.image.get_rect()
                self.rect.center = (x,y)

        def update(self,color):
                pygame.draw.circle(self.image,color,[15,15],15,0)
                pygame.draw.circle(self.image,black,[15,15],15,line_width)

#矢印ボタンの定義
class Triangle(pygame.sprite.Sprite):
```

```
                if dir == 1:
                        point_list = [[15,0],[30,30],[0,30]]
                elif dir == 2:
                        point_list = [[30,15],[0,30],[0,0]]
                elif dir == 3:
                        point_list = [[15,30],[0,0],[30,0]]
                else :
                        point_list = [[0,15],[30,0],[30,30]]
                pygame.draw.polygon(self.image,gray,point_list,0)
                pygame.draw.polygon(self.image,black,point_list,line_width)

                self.rect = self.image.get_rect()
                self.rect.center = (x,y)
                self.dir = dir

        def update(self,color):
                if self.dir == 1:
                        point_list = [[15,0],[30,30],[0,30]]
                elif self.dir == 2:
                        point_list = [[30,15],[0,30],[0,0]]
                elif self.dir == 3:
                        point_list = [[15,30],[0,0],[30,0]]
                else :
                        point_list = [[0,15],[30,0],[30,30]]
                pygame.draw.polygon(self.image,color,point_list,0)
                pygame.draw.polygon(self.image,black,point_list,line_width)

#L1,L2,R1,R2 ボタンの定義
class Rect(pygame.sprite.Sprite):
        def __init__(self,x,y):
                pygame.sprite.Sprite.__init__(self)
                self.image = pygame.Surface([50,20])
                self.image.fill(white)
                pygame.draw.rect(self.image,gray,[0,0,50,20],0)
                pygame.draw.rect(self.image,black,[0,0,50,20],line_width)
                self.rect = self.image.get_rect()
                self.rect.center = (x,y)

        def update(self,color):
```

```
                pygame.draw.rect(self.image,color,[0,0,50,20],0)
                pygame.draw.rect(self.image,black,[0,0,50,20],line_width)

#ジョイスティックの定義
class Stick(pygame.sprite.Sprite):
        def __init__(self,x,y):
                pygame.sprite.Sprite.__init__(self)
                self.image = pygame.Surface([100,100])
                self.image.fill(white)
                pygame.draw.rect(self.image,gray,[0,0,100,100],0)
                pygame.draw.rect(self.image,black,[0,0,100,100],line_width)
                pygame.draw.circle(self.image,white,[50,50],3,0)
                self.rect = self.image.get_rect()
                self.rect.center = (x,y)

        def update(self,x,y):
                pygame.draw.rect(self.image,gray,[0,0,100,100],0)
                pygame.draw.rect(self.image,black,[0,0,100,100],line_width)
                pygame.draw.circle(self.image,white,[x,y],3,0);

########################################################################
#ここから実行される
########################################################################
pygame.init()
#fps = pygame.time.Clock()
window = pygame.display.set_mode((600,300))                    #表示ウインドウの初期化

#ボタンなどの初期化
button1 = Circle(500,100)
button2 = Circle(550,150)
button3 = Circle(500,200)
button4 = Circle(450,150)

buttonU = Triangle(100,100,1)
buttonR = Triangle(150,150,2)
buttonD = Triangle(100,200,3)
buttonL = Triangle(50,150,4)

buttonL1 = Rect(100,30)
```

```
buttonL2 = Rect(100,60)
buttonR1 = Rect(500,30)
buttonR2 = Rect(500,60)

stick1 = Stick(200,230)
stick2 = Stick(400,230)

#センサ入力値を擬似カラー表示するための LUT
colormap_r = [255]*64 + range(255,0,-2) + [0]*64
colormap_g = range(0,255,4) + [255]*64 + range(255,0,-1)
colormap_b = [0]*128 + range(0,255,4) + [255]*64

xl,yl,xr,yr = 50,50,50,50                                   #ジョイスティックの位置を示す変数

while True:
        #データを 8 文字単位で処理する
        for c in file.read(1):
                data += ['%02X' % ord(c)]                   #文字 c をコードに変換し、それを文字化する
                if len(data) == 8:
                        if data[6] == '02':                 #ボタンが押し込まれる圧力を出力ありの場合
                                pos = int(data[4],16) + int(data[5],16)*256
                                if pos > 128*256:
                                        pos = -256*256 + pos
                                pos = pos+32767
                                i256 = pos/256              #センサ出力を 0～255 に変換
                                i100 = pos/655              #センサ出力を 0～100 に変換
                                color = colormap_r[i256],colormap_g[i256],colormap_b[i256]

                                if i256 == 0:               #センサ出力がゼロであれば、表示を初期化する
                                        button1.update(gray)
                                        button2.update(gray)
                                        button3.update(gray)
                                        button4.update(gray)
                                        buttonU.update(gray)
                                        buttonR.update(gray)
                                        buttonD.update(gray)
                                        buttonL.update(gray)
                                        buttonL1.update(gray)
```

```
            buttonL2.update(gray)
            buttonR1.update(gray)
            buttonR2.update(gray)
elif data[7] == '00':                   #左ジョイスティックの横移動
            xl = i100
            stick1.update(xl,yl)
elif data[7] == '01':                   #左ジョイスティックが縦移動
            yl = i100
            stick1.update(xl,yl)
elif data[7] == '02':                   #右ジョイスティックが横移動
            xr = i100
            stick2.update(xr,yr)
elif data[7] == '03':                   #右ジョイスティックが縦移動
            yr = i100
            stick2.update(xr,yr)
elif data[7] == '08':                   #上矢印キー
            buttonU.update(color)
elif data[7] == '09':                   #右矢印キー
            buttonR.update(color)
elif data[7] == '0A':                   #下矢印キー
            buttonD.update(color)
elif data[7] == '0B':                   #左矢印キー
            buttonL.update(color)
elif data[7] == '0C':                   #L2 キー
            buttonL1.update(color)
            sys.stdout.write('L2¥n')
elif data[7] == '0D':                   #R2 キー
            buttonR1.update(color)
            sys.stdout.write('R2¥n')
elif data[7] == '0E':                   #L1 キー
            buttonL2.update(color)
            sys.stdout.write('L1¥n')
elif data[7] == '0F':                   #R1 キー
            buttonR2.update(color)
            sys.stdout.write('R1¥n')
elif data[7] == '10':                   #△ボタン
            button1.update(color)
elif data[7] == '11':                   #○ボタン
```

```
                button2.update(color)
        elif data[7] == '12':                   #×ボタン
                button3.update(color)
        elif data[7] == '13':                   #□ボタン
                button4.update(color)

#ウィンドウの表示を更新する
window.fill(white)
window.blit(button1.image,button1.rect)
window.blit(button2.image,button2.rect)
window.blit(button3.image,button3.rect)
window.blit(button4.image,button4.rect)
window.blit(buttonU.image,buttonU.rect)
window.blit(buttonR.image,buttonR.rect)
window.blit(buttonD.image,buttonD.rect)
window.blit(buttonL.image,buttonL.rect)
window.blit(buttonL1.image,buttonL1.rect)
window.blit(buttonL2.image,buttonL2.rect)
window.blit(buttonR1.image,buttonR1.rect)
window.blit(buttonR2.image,buttonR2.rect)
window.blit(stick1.image,stick1.rect)
window.blit(stick2.image,stick2.rect)
pygame.display.update()

#バッファを初期化する
data = []
```

付録 2: カメラモジュールで顔を検出するとモータを止めるサンプルコード

```
# -*- coding: utf-8 -*-
# PS リモコンの信号を受け取り、DC モータを駆動させて、顔検出を開始する。
#  顔を検出した場合、DC モータの駆動を止める信号を出力し、止める。
import cv2
import io
import picamera
import numpy as np
import time
import sys
import RPi.GPIO as GPIO

# GPIO を初期化する
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
GPIO.output(17, GPIO.LOW)
GPIO.output(22, GPIO.LOW)
#    picamera モジュールの機能を使う方法で、
# picamera の設定を while ループの外で行う
# camera.capture で use_video_port = True とする（= False より画質が悪い）
# format は jpeg 以外に、png なども可能であるが、jpeg の方がが高速
cv2.namedWindow('kao',cv2.WINDOW_AUTOSIZE)                    # カメラのウインドウを用意する
camera = picamera.PiCamera()                                  # カメラを起動する
camera.resolution = (160,120)                                 # 表示画像の大きさ　160×120
cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")  # カースケード検出器を呼び出す。
# DS3 用リモコンの信号を入力するためのデバイスファイルをオープンする
```

```
file = open('/dev/input/js0','r')
data=[]                                                                 # リモコンからの信号を読み込むための変数
flag = 1                                                                # 初期値　flag = 1

while flag:                                                             # while 文を回す。
        for character in file.read(1):                                  # 1 バイト読み込む
                data += ['%02X' % ord(character)]                       # 読み込んだ 1 バイトをコードに変換し、それを文字化する
                if len(data) == 8:                                      # リモコンからの信号は 8 バイト単位になっている
                        if data[6] == '01':                             # XXX の場合
                                if data[4] == '01':                     # ボタンが押された場合
                                        if data[7] == '0C':             # △ボタンが押された場合
                                                GPIO.output(17, GPIO.HIGH) # GPIO-17 を HIGH にする
                                                GPIO.output(22, GPIO.HIGH) # GPIO-22 を HIGH にする
                                                flag = 0                # flag を 0 にすることで強制的に抜ける
                        data = []                                       # リモコンからの信号を読み込むための変数
while 1:
        stream = io.BytesIO()                                           # ストリーム処理する
        camera.capture(stream,format = 'jpeg',use_video_port = True)    # フォーマットを jpeg に設定し、ポートを True とする
        cap = np.fromstring(stream.getvalue(), dtype=np.uint8)          # データ型の配列を np.utit8
        im = cv2.imdecode(cap,0)                                        # メモリバッファに画像を読み込む
        face = cascade.detectMultiScale(im, 1.1, 3)                     # 入力画像中の異なるサイズの物体を検出する
        for (x, y, w, h) in face:                                       # 顔検出
                cv2.rectangle(im,(x,y),(x+w,y+h),(0,50,255),3)          # 顔検出した場合顔の周りを線で囲う
                GPIO.output(17, GPIO.LOW)                               # GPIO-17 を LOW にする
                GPIO.output(22, GPIO.LOW)                               # GPIO-22 を LOW にする
        cv2.imshow('kao',im)                                            # 表示させる
        cv2.waitKey(1)                                                  # 表示させている画面を維持する
```

付録 3：サーボモータを制御するサンプルコード

```
# -*- coding: utf-8 -*-
# PS3 用リモコン(Dual Shock 3)を用いて、サーボモーターを制御する
#
import sys
import os

# DS3 用リモコンの信号を入力するためのデバイスファイルをオープンする
file = open('/dev/input/js0','r')

#リモコンからの信号を読み込むための変数
data = []

#初期値
servo_angle = 150
angle_max = 200
angle_min = 100
inc = 5

while 1:
        for c in file.read(1):                                          # 1 バイト読み込む
                data += ['%02X' % ord(c)]                               #  読み込んだ 1 バイトをコードに変換し、それを文字化する
                if len(data) == 8:                                      #  リモコンからの信号は 8 バイト単位になっている
                        if data[6] == '02':                             # XXX の場合
                                if data[7] == '0F' :                    # R1 ボタンを押した場合
                                        if (servo_angle-inc) >= angle_min:      # servo_angle が angle_min より小さくない場合
                                                servo_angle = servo_angle-inc  # servo_angle から-inc する
```

```
                    # サーボブラスター(デバイスファイル)に値を送る。2 は GPIO-18 を使うことを意味する。
                    cmd = 'echo 2=' + str(servo_angle) + '> /dev/servoblaster'
                    sys.stdout.write(cmd +'¥n')     # cmd の値を表示する
                    os.system(cmd)                  # サブシェル内でコマンド(文字列)を実行する
            elif data[7] == '0E':                   # L1 ボタンを押した場合
                if (servo_angle+inc) <= angle_max:  # servo_angle が angle_max より大きくない場合
                    servo_angle = servo_angle+inc # servo_angle から+inc する

                    # サーボブラスター(デバイスファイル)に値を送る。2 は GPIO-18 を使うことを意味する。
                    cmd = 'echo 2=' + str(servo_angle) + ' > /dev/servoblaster'
                    sys.stdout.write(cmd +'¥n')     # cmd の値を表示する
                    os.system(cmd)                  # サブシェル内でコマンド(文字列)を実行する
            elif data[7] == '10':                   # 三角ボタンを押した場合

                    # サーボブラスター(デバイスファイル)に値を送る。
                    cmd = 'echo 2=150 > /dev/servoblaster'
                    servo_angle = 150                       # 初期値に戻す
                    os.system(cmd)                          # サブシェル内でコマンド(文字列)を実行する
sys.stdout.flush()                                          # コンソールへの出力を上書きしてゆく方法
data = []                                                   # リモコンからの信号を読み込むための変数
```

付録 4：リモコンで模型自動車を制御するサンプルコード

```
# -*- coding: utf-8 -*-
# PS3 用リモコン(Dual Shock 3)を用いて、サーボモーターを制御する
#
import sys
import os
import RPi.GPIO as GPIO

#GPIO を初期化する。サーボブラスターが使用するポートを避ける。
GPIO.setmode(GPIO.BCM)
GPIO.setup(14, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.output(14, GPIO.LOW)
GPIO.output(15, GPIO.LOW)

# DS3 用リモコンの信号を入力するためのデバイスファイルをオープンする
file = open('/dev/input/js0','r')

#リモコンからの信号を読み込むための変数
data = []

#初期値
servo_angle = 150
angle_max = 200
angle_min = 100
inc = 5
```

```
while 1:
        for c in file.read(1):                                          # 1 バイト読み込む
                data += ['%02X' % ord(c)]                               # 読み込んだ 1 バイトをコードに変換し、それを文字化する
                if len(data) == 8:                                      # リモコンからの信号は 8 バイト単位になっている
                        if data[6] == '02':                             # ボタン圧力をセンスするモード
                                #sys.stdout.write(data[7])
                                if data[7] == '0E' :                    # L1 ボタンを押した場合
                                        if (servo_angle-inc) >= angle_min:      # servo_angle が angle_min より小さくない場合
                                                servo_angle = servo_angle-inc  # servo_angle から-inc する

                                                # サーボブラスター(デバイスファイル)に値を送る。0 は GPIO-4 を使うことを意味する。
                                                cmd = 'echo 0=' + str(servo_angle) + '> /dev/servoblaster'
                                                sys.stdout.write(cmd +'¥n')     # cmd の値を表示する
                                                os.system(cmd)                  # サブシェル内でコマンド(文字列)を実行する
                                elif data[7] == '0C':                   # L2 ボタンを押した場合
                                        if (servo_angle+inc) <= angle_max:      # servo_angle が angle_max より大きくない場合
                                                servo_angle = servo_angle+inc # servo_angle から+inc する

                                                # サーボブラスター(デバイスファイル)に値を送る。0 は GPIO-4 を使うことを意味する。
                                                cmd = 'echo 0=' + str(servo_angle) + ' > /dev/servoblaster'
                                                sys.stdout.write(cmd +'¥n')     # cmd の値を表示する
                                                os.system(cmd)                  # サブシェル内でコマンド(文字列)を実行する
                                elif data[7] == '08':                   # ↑ボタンを押した場合
                                                # サーボーブラスター(ディバイスファイル)に値を送る。
                                                cmd = 'echo 4=150 > /dev/servoblaster'
                                                servo_angle = 150               # 初期値に戻す
                                                os.system(cmd)                  # サブシェル内でコマンド(文字列)を実行する
```

```
elif data[6] == '01':                                   #ボタンのオン・オフを検出するモード
        if data[4] == '01':                             #ボタンが押された場合
                if data[7] == '0B':                     # R1 ボタンダウン、前進
                        GPIO.output(14, GPIO.HIGH)
                        GPIO.output(15, GPIO.LOW)
                        sys.stdout.write('R1 down¥n')
                elif data[7] == '09':                   # R2 ボタンダウン、後退
                        GPIO.output(14, GPIO.LOW)
                        GPIO.output(15, GPIO.HIGH)
                        sys.stdout.write('R2 down¥n')
        elif data[4] == '00':                           # ボタンが離された場合
                if data[7] == '0B':                     # R1 ボタンアップ、ストップ
                        GPIO.output(14, GPIO.LOW)
                        GPIO.output(15, GPIO.LOW)
                        sys.stdout.write('R1 up¥n')
                elif data[7] == '09':                   # R2 ボタンアップ、ストップ
                        GPIO.output(14, GPIO.LOW)
                        GPIO.output(15, GPIO.LOW)
                        sys.stdout.write('R2 up¥n')

sys.stdout.flush()                                      # コンソールへの出力を上書きしてゆく方法
data = []                                               # リモコンからの信号を読み込むための変数
```