

Raspberry Pi による制御

名古屋大学情報学部コンピュータ科学科 3 年

堀内颯太 (学籍番号 : 101730331)

horiuchi.sota@e mbox.nagoya-u.ac.jp

実験日 2019 / 4 / 25

目次

| | | |
|-----|---|----|
| 1 | はじめに | 3 |
| 1.1 | 概要 | 3 |
| 1.2 | 実験で使用する器具について | 3 |
| 2 | 課題 1: Raspberry pi による単一 LED (Light Emitting Diode) の点滅制御 | 5 |
| 2.1 | 目的・概要 | 5 |
| 2.2 | 実験方法 | 5 |
| 2.3 | 実験結果 | 7 |
| 2.4 | 考察 | 7 |
| 3 | 課題 2: Raspberry pi による单ースイッチの状態読み取り | 7 |
| 3.1 | 目的・概要 | 7 |
| 3.2 | 実験方法 | 8 |
| 3.3 | 実験結果 | 9 |
| 3.4 | 考察 | 10 |
| 4 | 調査課題 2-1: チャタリングについて | 12 |
| 4.1 | ハードウェア的方法 | 12 |
| 4.2 | ソフトウェア的方法 | 13 |
| 5 | 調査課題 2-2: プルアップ及びプルダウンについて | 14 |
| 6 | 課題 3: Raspberry pi による複数 LED とスイッチの制御 | 16 |
| 6.1 | 目的・概要 | 16 |
| 6.2 | 実験方法 | 16 |
| 6.3 | 実験結果 | 21 |
| 6.4 | 考察 | 21 |
| 7 | 課題 4: Raspberry pi による单一ステッピングモータの制御 | 23 |
| 7.1 | 目的・概要 | 23 |
| 7.2 | 実験方法 | 23 |
| 7.3 | 実験結果 | 29 |
| 7.4 | 考察 | 29 |
| 8 | 課題 5: Raspberry pi による複数ステッピングモータの制御 | 29 |
| 8.1 | 目的・概要 | 29 |
| 8.2 | 実験方法 | 29 |
| 8.3 | 実験結果 | 33 |
| 8.4 | 考察 | 33 |

| | | |
|-----|----------------------------------|----|
| 9 | 課題 6: マニュアル操作ステッピングモータカーの作成 | 35 |
| 9.1 | 目的・概要 | 35 |
| 9.2 | 実験方法 | 35 |
| 9.3 | 実験結果 | 41 |
| 9.4 | 考察 | 42 |
| 10 | 調査課題 6-1 Zeroconf の代表的なプロトコルについて | 42 |
| 11 | まとめ | 43 |

1 はじめに

1.1 概要

本レポートでは小型コンピュータの一つである Raspberry pi を用いて、電子制御の実験を行う。まず、Raspberry pi による制御及び GPIO 端子の接続、ブレッドボードの利用方法理解するため単一 LED の点滅プログラムを作成し制御を行う（実験 1）。次にスイッチの状態読み取りを理解するために单ースイッチの状態の読み取り動作の確認を行う（実験 2）。そして、スイッチを任意のタイミングで押し複数の LED の点灯及び消灯を行う制御を理解する（実験 3）。レポートの後半ではステッピングモータの 1 相、2 相、1-2 相励磁駆動方式を Raspberry pi を用いて実現する（実験 4）。その後スイッチ操作により 2 つのステッピングモータの正転、逆転、停止の制御を行う（実験 5）。そして、ステッピングモータのスイッチ操作によるマニュアル操作駆動ステッピングモータカバーの作成を行う（実験 6）。最後に本レポートで述べるすべての実験に対しての総括をし、Raspberry pi による制御についてまとめ、電子制御に関する理解を深める。

1.2 実験で使用する器具について

1.2.1 Raspberry pi

Raspberry pi はラズベリーパイ財団により開発された安価な教育用のシングルボードコンピューターである。内蔵ハードディスクなどを搭載せず、電源や SD カードストレージを装着することにより使用可能な、ワンボードマイコンと呼ばれるハードウェアの一つである。GPU が搭載されており、グラフィック分析や処理が可能なほか HDMI による出力や、USB ポート、LAN ポート、OS が使用可能であり、通常のパソコンと同等の機能を持つ。Raspberry pi は GPIO(General Purpose Input/Output) 端子を有しており、ここに接続することで電子部品の制御が容易に行うことが可能である。これを利用して Raspberrypi を組み込むことで様々な機器の IoT 化を可能にすることができる。[1]

本実験では Raspberry pi の OS は Raspberry pi での電子制御を可能にするソフトウェアが入った Raspbian を使用する。また、GPIO を使用するために、GPIO 操作ライブラリである WiringPi を使用する。wiringpi は C 言語ライブラリであるが、ラッパが用意されており、Python といった他の言語でもこのライブラリにアクセスできる。そのため今回の制御プログラムの言語には Python3 を用いる。wiringpi のインストールは以下のコマンドをターミナルに打ち込むことで行うことができる。

```
pip3 install wiringpi
```

1.2.2 GPIO 端子

GPIO は Raspberry pi が電子部品に接続する際に使用する。GPIO は図 1 のように 40 本のピンをもつ。GPIO の電源端子は同じ役割のものが複数あるため、接続しやすいピンを使用する。しかし、GPIO 端子が小さく、配線に不便である。そのため配線を容易に行うことができるようブレッドボードを使用する。

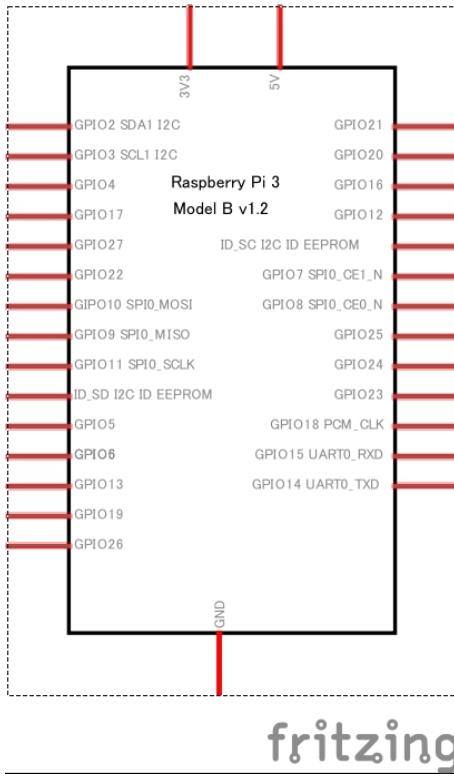


図 1 Raspberry pi のピン

1.2.3 ブレッドボード

ブレッドボードは電子回路を簡易的に作るための基板である。はんだ付けなどが不要で、ボードに部品を差し込むことで、部品同士を接続させ、安易に電子回路を組むことが可能である。^[2] ブレッドボードは内部で接続されている箇所と接続されていない箇所がある。図 2 はブレッドボードである。上下の-,+ は横のラインで穴が接続されており中央は縦のラインでは 1 ラインにつき横の A、B、C、D と E、F、G、H で分かれておりそれぞれ内部で導通している。実験では Raspberry pi の GPIO とブレッドボードを接続するためにパラレルケーブルと T 字型基盤を用いる。T 字型ケーブルのピン配置は図 3 であり、図のように接続する。パラレルケーブルの片方は T 字型基盤の左側のブレッドボードに接続しない部分、もう一方を GPIO 端子に接続することで配線の作成を容易にすることが可能である。

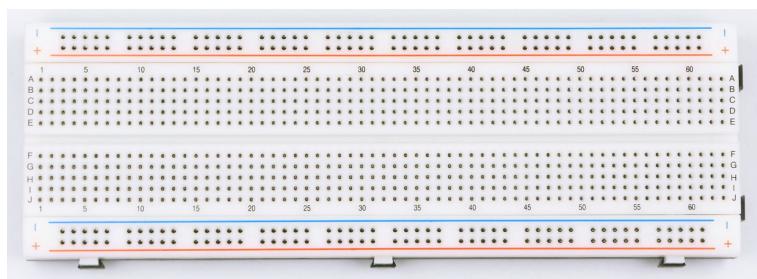


図 2 ブレッドボード

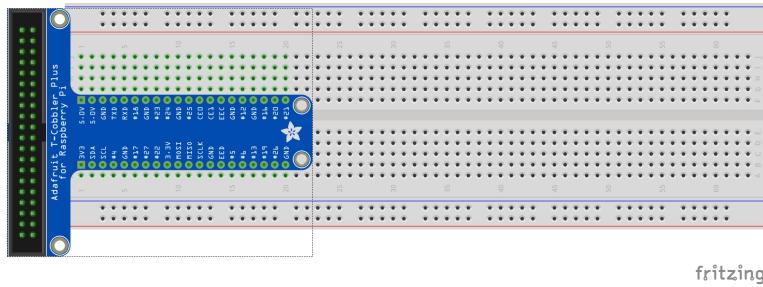


図 3 T 字型基盤

2 課題 1: Raspberry pi による単一 LED (Light Emitting Diode) の点滅制御

2.1 目的・概要

本実験ではマイコンである Raspberry pi で LED の点滅プログラムの作成及び実行をした。

2.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- Raspberry Pi 3 Model B v1.2 (CPU:1.2GHz)
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- HDMI ケーブル
- USB キーボード及びマウス
- Micro USB 電源ケーブル
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- LED 及び抵抗を実装した基盤

下準備として、Raspberry pi に Micro SD カード、USB キーボード及びマウス、Micro USB 電源ケーブルを接続し、Raspberry pi で電子制御を行えるようにした。また HDMI ケーブルを用いて Raspberry pi とディスプレイを接続した。

まず、Raspberrypi の GPIO に GPIO ブレッドボード接続ケーブルの T 字型基盤が接続していない側を接続した。T 字型基盤をブレッドボードに接続した。ブレッドボードを使い、図 4 のように回路を作成した。LED の抵抗値が低く回路に電流が流れすぎるのを防ぐため、抵抗を直列に付けた。抵抗側を GPIO ピンに、LED 側を GND ピンに接続した。ソースコード 1 のプログラムコードを実行することで LED の点滅の実行を行った。実行コマンドは python3 led.py で行った。LED の点滅の様子を動画にとり、LED の点灯が始まった直後から 3 回目の点灯が終わるまでの時間を測定し、その時間を 5 で割ることで LED の点滅の間隔としこれを 10 度繰り返して点滅の間隔を測定した。

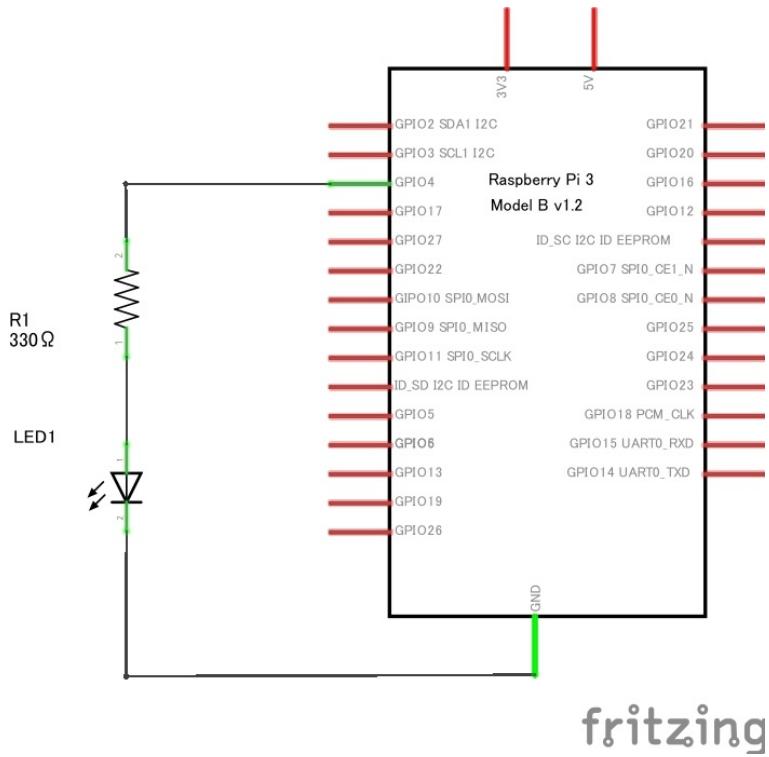


図4 Rasberry pi と単一 LED を接続する回路図

R:抵抗 LED:Light Emitting Diode GND:グランド

ソースコード 1: led.py

```

1 import wiringpi as pi, time
2
3 #LED 接続 GPIO 端子番号
4 LED_PIN = 4
5
6 #LED 接続端子を出力モードにする
7 pi.wiringPiSetupGpio()
8 pi.pinMode(LED_PIN, pi.OUTPUT)
9
10 #LED を点滅させる
11 #プログラムが停止するまで半永久的に High と Low を繰り返す
12 #time.sleep を入れて LED の点滅間隔を作る
13 while True:
14     #LED 接続端子を High にする
15     pi.digitalWrite(LED_PIN, pi.HIGH)
16     #点滅のために一時的にプログラムを止める
17     time.sleep(1)

```

```

18
19 #LED 接続端子を Low にする
20 pi.digitalWrite(LED_PIN, pi.LOW)
21 #点滅のために一時的にプログラムを止める
22 time.sleep(1)

```

2.3 実験結果

LED の点滅間隔は表 1 の通りとなった。LED 点灯している時間はおよそ 1 秒程度もしくは 1.005 秒程度であった。

表 1 LED の点滅間隔

| | |
|------|--|
| 標本数 | 10 |
| 標本 | 0.994 1.009 0.992 1.062 1.014 1.0 1.004 0.991 1.001 0.991 |
| 平均 | 1.0058 |
| 不变分散 | 0.0004515111 |
| 標準偏差 | 0.0212487908 |
| 誤差 | 0.00671945765 |
| 測定値 | 1.0058 ± 0.00671945765 |

2.4 考察

LED の点滅間隔について考える。予想される点滅間隔は 1 秒であり、これはプログラム内の `time.sleep(1)` により High になっている間隔と Low になっている間がともに 1 秒だからである。しかし、測定値ちょうど 1 秒値にならず、やや多い値になった。これは `pi.digitalWrite(LED_PIN, pi.HIGH)` 及び `pi.digitalWrite(LED_PIN, pi.LOW)` の実行にやや多くなったその時間がかかたためであると考えられる。Raspberry Pi の CPU 速度は 1.2GHz であるため 1 秒間に 1.2×10^9 回の処理を行いその一回の処理の間隔は 8.33×10^{-10} である。誤差を考慮すると最大 0.012595 秒の時間が `time.sleep(1)` 以外で余分にかかっていることになりこれが `pi.digitalWrite()` の実行時間の上限である。つまり、`pi.digitalWrite()` は最大約 1.5×10^7 回 CPU で処理を行う可能性があることが分かった。ただし、この計測時間は肉眼での計測である以上正確な時間を計測できたとは言えず、もっと少ない処理回数であると考えられる。

3 課題 2: Raspberry pi による单ースイッチの状態読み取り

3.1 目的・概要

本実験では、Raspberry Pi の電子制御の一つとして单ースイッチの読み取りを行うプログラムを作成及び実行を行った。

3.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- Raspberry Pi 3 Model B v1.2 (CPU:1.2GHz)
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- HDMI ケーブル
- USB キーボード及びマウス
- Micro USB 電源ケーブル
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- 抵抗 (1k Ω)
- 押しボタンスイッチ

下準備として実験 1 同様、Raspberry pi に Micro SD カード、USB キーボード及びマウス、Micro USB 電源ケーブルを接続し、Raspberry pi で電子制御を行えるようにした。また HDMI ケーブルを用いて Raspberry pi とディスプレイを接続し、また、Raspberrypi の GPIO に GPIO ブレッドボード接続ケーブルの T 字型基盤が接続していない側を接続した。T 字型基盤をブレッドボードに接続した。ブレッドボードを使い、図 5 のように回路を作成した。GPIO のピンに抵抗を接続し、スイッチ側は GND に接続した。ソースコード 2 のプログラムコードを実行することでスイッチの状態読み取りを行った。実行コマンドは python3 sw.py で行った。スイッチを押している際は Switch On を、押されていない場合は Switch off を 0.5 秒間隔で端末に表示させた。

ソースコード 2: sw.py

```
1 import wiringpi as pi, time
2
3 #スイッチ接続 GPIO 端子番号
4 SW_PIN = 4
5
6 #スイッチ接続端子を入力モードにする
7 pi.wiringPiSetupGpio()
8 pi.pinMode(SW_PIN, pi.INPUT)
9 #スイッチ接続端子をプルアップモードにする
10 pi.pullUpDnControl(SW_PIN, pi.PUD_UP)
11
12
13 while True:
```

```

14 #スイッチ接続端子の状態読み取り
15 #スイッチが押されていれば Switch on そうでないなら Off を
16 if (pi.digitalRead(SW_PIN) == 0):
17     print("Switch on")
18 else:
19     print("Switch off")
20
21 time.sleep(0.5)

```

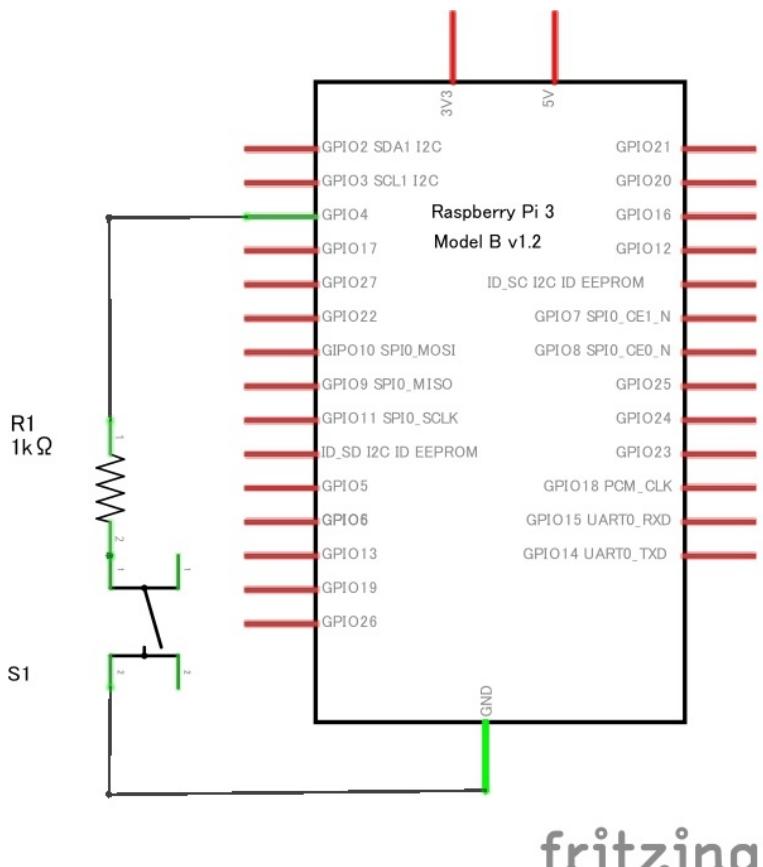


図 5 Rasberry pi と單一スイッチを接続する回路図

R:抵抗 s:スイッチ GND:グラウンド

3.3 実験結果

スイッチを押すと Switch on が表示された。スイッチを押していないときは Switch off が表示された。スイッチを押しておらず Switch off が表示され、次の文字が表示される前にスイッチを押し、離すという行為を行うと次点で表示される文字は Switch off であった。また、スイッチを押しているときに Switch on が表示され、次の文字が表示される前にスイッチを離し直後に押しつづけた場合次に表示される文字は Switch off

であった。

3.4 考察

本実験のプログラムのフローチャートは図 6 のとおりである。端末に表示する on off の判断材料となるスイッチの状態の読み取りは状態読み取りのときにしか行われずそれまでにスイッチが押されたかどうかを読み取ることはできないため、状態が表示されたのち次の状態が表示されるまでの 0.5 秒間にスイッチの状態を変化させ戻すという行為を行っても表示される文字に影響を与えたかったと考えられる。つまり、今回の実験のプログラムでは 0.5 秒間隔でのスイッチの状態の読み取りを行うことはできたが、スイッチを押された時にスイッチが押されているということを認識し、画面に出力することはできないと考えられる。スイッチの On が行われたかどうかを判別するには例えばソースコード 2-1:new_sw.py のようなプログラムを実行し同じようにスイッチを押す動作をすればスイッチを押したときに Switch on が表示されるようになると考えられる。この場合 0.01 秒間隔でスイッチの読み取りを行いスイッチが押された場合 Switch on が表示され、1 秒間スイッチが押されない場合は Switch off を表示する。スイッチが押された場合はまた for のループの外からやり直しすることで 1 秒間スイッチが押されないという判定になる。しかしこの場合 0.01 秒間以上スイッチが押されていると Switch on が表示され続けるので注意が必要である。

ソースコード 2-1: new_sw.py

```
1 import wiringpi as pi, time
2
3 #スイッチ接続 GPIO 端子番号
4 SW_PIN = 4
5
6 #スイッチ接続端子を入力モードにする
7 pi.wiringPiSetupGpio()
8 pi.pinMode(SW_PIN, pi.INPUT)
9 #スイッチ接続端子をプルアップモードにする
10 pi.pullUpDnControl(SW_PIN, pi.PUD_UP)
11
12
13 while True:
14     for i in range(10**2):
15         #スイッチ接続端子の状態読み取り
16         #もしスイッチが押されたら print
17         if (pi.digitalRead(SW_PIN) == 0):
18             print("Switch on")
19             break
20         time.sleep(0.01)
21     else:
```

```
print("Switch off")
```

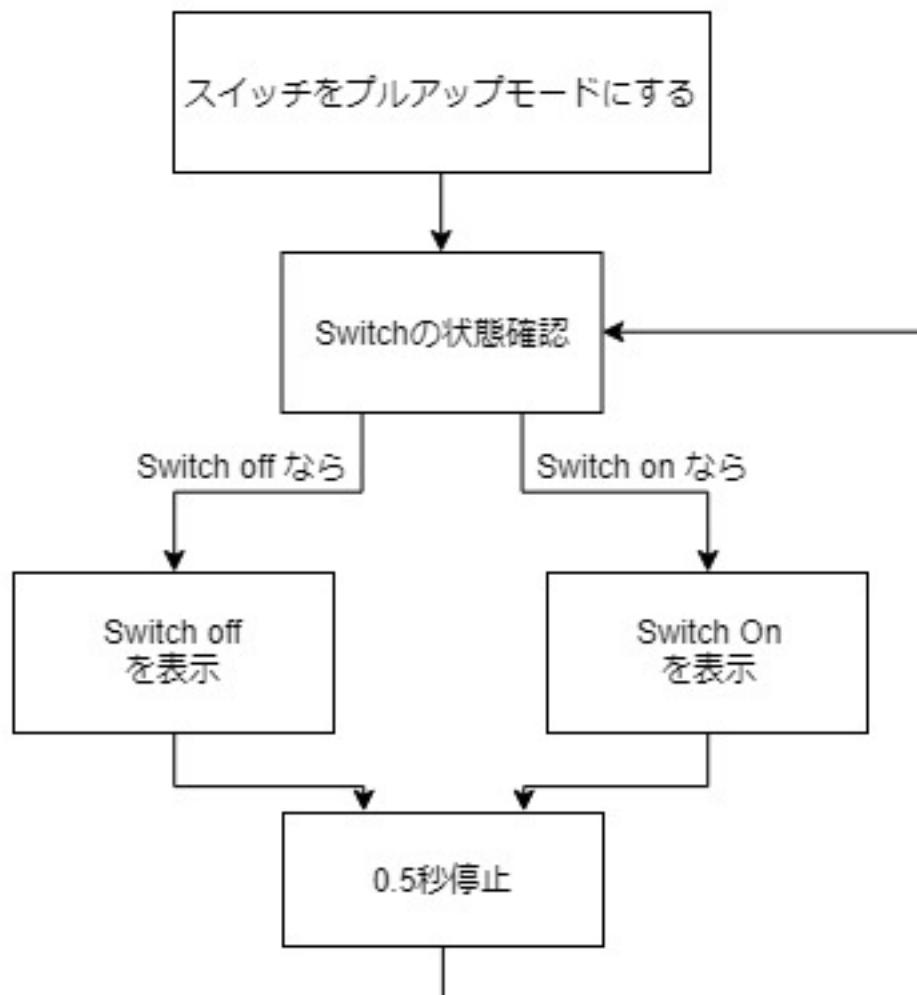


図 6 実験 2 のフローチャート

4 調査課題 2-1: チャタリングについて

トグルスイッチ、押しボタンスイッチといった機械式スイッチではチャタリングという現象が起こる。チャタリングとはオンされている接点が外部の力により開閉を反復すること非常に短い時間に複数の電子パルスが発生する現象であり、具体的にはスイッチ ON/OFF 直後に出力が短い時間 ON/OFF を繰り返されて起こる現象である。スイッチの種類にもよるが、数百マイクロ秒から数十ミリ秒程度の間に、数回から十数回の ON/OFF が発生する。ON または OFF 時に接点が 1 度で接続、または 1 度で接点が離れる理想的なスイッチであれば、チャタリングは発生しない。しかし、実際のスイッチは接点のバウンドまたは擦れが発生するため、これによりチャタリングが起こる。チャタリングを軽減する方法はハードウェア的方法及びソフトウェア的方法双方に存在する。[3]

4.1 ハードウェア的方法

ハードウェア的方法でチャタリングを軽減する一つは CR の充放電を利用する方法がある。この方法は図 7 のように抵抗とコンデンサによる充放電を利用したものであり A の点でスイッチの状態を読み取る。十分な時間スイッチが Off になっているとき B の電圧は High である。ここからスイッチを押す場合、B の電圧はスイッチ経由で GND に放電され、GND まで下がり Low になる。この放電時間がチャタリング時間より十分長ければ、B 点の電圧の波形はチャタリングの影響を受けず図 8 のようになる。一方、十分な時間スイッチが On になっているときの B の電圧は Low(GND) である。ここからスイッチを離す場合、B の電圧は Vcc から充電され、Vcc まで上がり Hihg になる。この充電時間がチャタリング時間より十分長ければ、点 B の電圧の波形はチャタリングの影響を受けず図 9 のようになる。このようにスイッチ ON/OFF 時にゆるやかな波形となつたものがインバーターに入力される。図 10 のように点 C で電圧が減少を繰り返し始めると点 A、B で放電が起り初め、徐々に電圧が下がり、点 B で L の認識レベルとなった時点で点 A の NOT 出力はチャタリングの無い H レベルになる。点 C で電圧の上昇を繰り返し始めると点 A、B で充電が始まり電圧が徐々に上がり、点 B で H の認識レベルとなった時点で点 C の NOT 出力は L レベルになる。NOT 出力はそれぞれの充放電時間遅れるものの、チャタリングを除去することが可能である。[3]

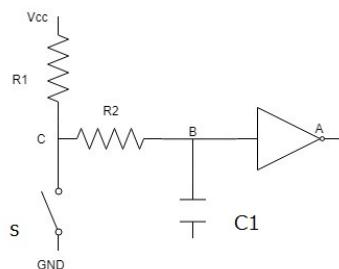


図 7 チャタリングのハードウェア的軽減方法回路 Vcc:電源電圧又は GPIO 端子電圧 GND:グランド C1:コンデンサー R1,R2:抵抗

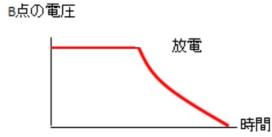


図 8 チャタリングのハードウェア的軽減方法
回路における点 B のスイッチオン後の電圧波形

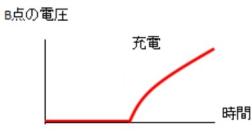


図 9 チャタリングのハードウェア的軽減方法
回路における点 B のスイッチオフ後の電圧波形

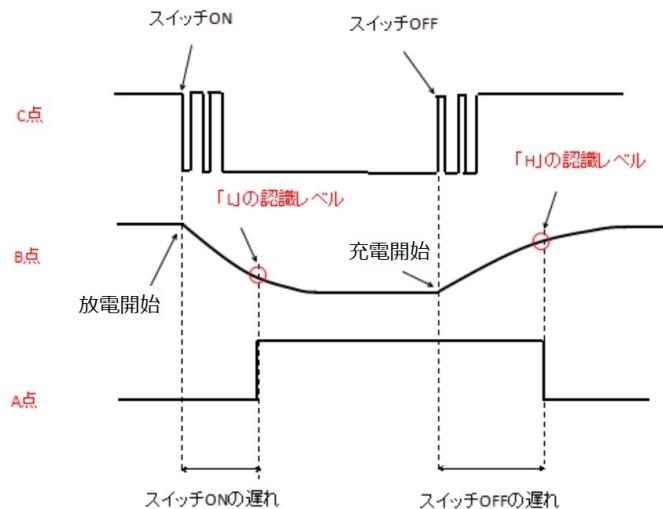


図 10 チャタリングのハードウェア的軽減方法回路における点 A,B,C のスイッチ動作の電圧波形 [3]

この方法はチャタリングが非常に短い時間でしか起こらないという性質を利用しておおり、端的に言えば一定時間以上スイッチ変化することでそのスイッチの変化を認める形でチャタリングを防いでいる。デメリットとしてはごくわずかな時間であるがスイッチの変化の認識に遅延が発生することと、本来ならば必要のないコンデンサーやインバーターを使うために部品数が多くなる点がある。また、一般のインバーターではゆるやかな信号を入力すると、H、L の認識レベル付近で誤動作してしまう。この誤作動を防ぐために、シュミット・トリガ・NOT を用いる。シュミット・トリガ・NOT はゆるやかな信号でも誤作動することがなく、波形整形などの用途で用いられる NOT IC である。遅れ時間は、正確には IC の種類、電源電圧、R1（抵抗値）、R2（抵抗値）、C1（コンデンサーの電気容量）の値で決まる。遅れ時間の目安の計算は $Time_{on} = R2 \times C1$ 、 $Time_{off} = (R1 + R2) \times C1$ ができる。[3]

4.2 ソフトウェア的方法

ソフトウェア的チャタリング軽減方法の一つにサンプリング+ゲージ判定方式が存在する。これはサンプリング方式の欠点をゲージ判定法により補う方法である。サンプリング方式はタイマ割り込みを使い一定時間ごとにスイッチの状態（ON/OFF）を読み出すというもので、複数スイッチの入力に対応することができる。この方式では、周期が訪れたタイミングで全てのスイッチの状態をメモリへ書き込み、プログラムはメモリ上のスイッチの状態を取得する。そして、次のスイッチの状態の読み出し時までに処理系は同じ値を読み取

り、安定した信号になり、チャタリング時間よりも長いサンプリング周期を取れば理論上はチャタリングが起こらない方法である。しかし、この方法は外来ノイズに弱く、ノイズが入って一瞬だけ ON になったタイミングに運悪く周期が差し掛かると、入力を受け付けて ON の入力として認識してしまう。この欠点をゲージ判定方式を取り入れることで補うのがサンプリング+ゲージ判定方式である。ゲージ判定法式は、スイッチごとに用意したカウンタに、スイッチが押されたたびそのスイッチに対応するカウンタをカウントアップし、ある一定回数以上の値（一致検出回数）に達したら入力を受けつけるという方式の判定方法である。この方式をサンプリング方式に組み合わせ、スイッチごとにカウンタを用意し、タイマ割り込みにより一定時間ごとにスイッチの状態（ON/OFF）を読み出し、ON であればそのカウンタをカウントアップし、ある一定以上の値（一致検出回数）に達したら入力を受け付けるという方式にすることで、カウントアップ中は入力として受け取らないため、チャタリング時間を超えるディレイを取って入力を受け取ればチャタリングを防ぐことができ、サンプリング方式の欠点を補うことができる。

チャタリングの防止の際、遅延を増やすほどチャタリングが起こりにくくなる。そのため、どの頻度までチャタリングを許し、どの程度まで遅延を許すかを考えることがソフトウェア、ハードウェアどちらにも求められる。

5 調査課題 2-2: プルアップ及びプルダウンについて

スイッチのオン/オフ状態を認識する場合、取得する電圧を安定化させることが必要である。そもそも、デジタル回路の信号としての電圧は、Hi または Low の電圧が常に印加されている必要がある。なぜなら、Hi と Low の中間電圧の状態では、入力を受け付ける側で Hi と Low をどの程度の電圧で判断するかが曖昧になり、誤動作を起こす事がある。他にも、回路の入力端子がどこにも接続されていないような状態の際、周囲の静電気や電磁誘導により電流が侵入し、高電圧が印加される場合がある。そのため、スイッチで読み取る端子部分に電圧を印加するか GND で一定にさせるかどちらかで電圧を安定化させる。印加する前者をプルアップ、GND にする後者をプルダウンという。どちらも同程度の能力を有しておりどちらが優れているということはないが、断線故障した時に Low(0V) の状態にしておくのが良いのか、High(この場合は 5V) にしておくのが良いのかで選択する。Low にしておきたい場合はプルダウン、High の場合はプルアップを選択する。

プルアップは図 11 のように、電源側に抵抗を接続させる事で、スイッチ OFF 時に A の電圧を吊り上げる。スイッチがオンの際は IN の電圧はおよそ 5V で固定され、ON の際はおよそ 0V となる。そのためプルアップでスイッチ操作を行うときの入力される論理値は、スイッチ入力があるときは 0 をスイッチ入力がない時は 1 になる。

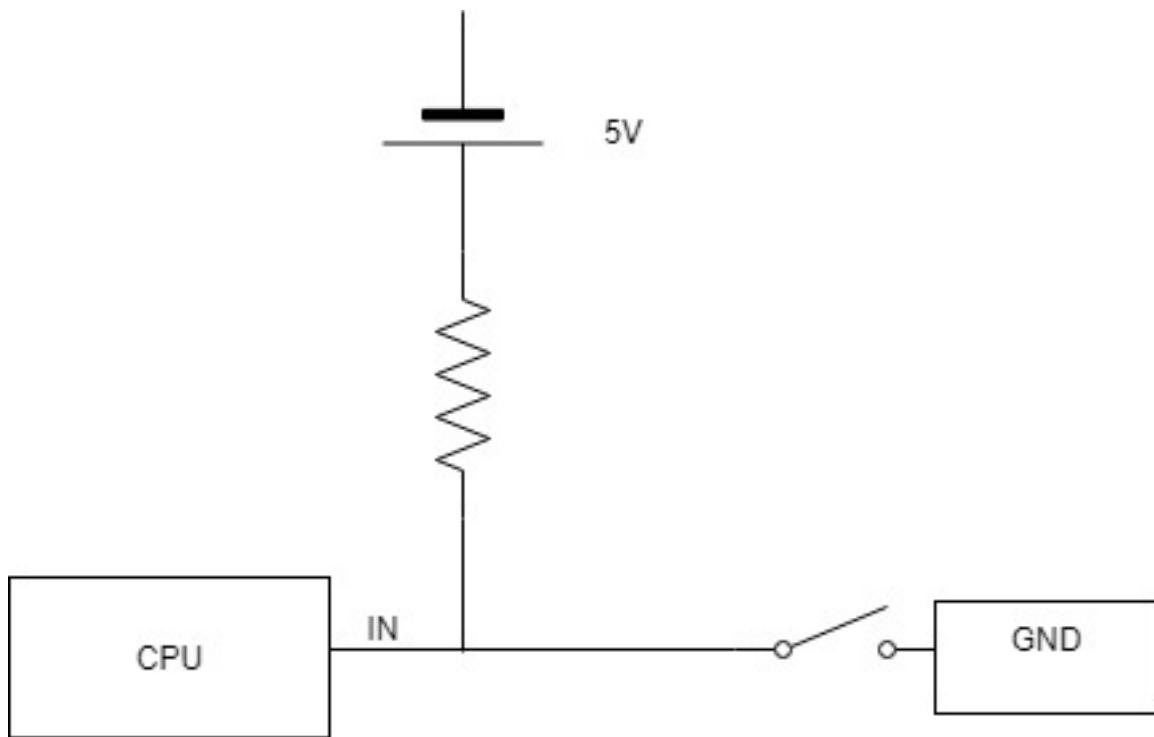


図 11 プルアップ回路図

一方、プルダウンは図 12 のように、プルアップと異なり、GND 側に接続し、電源→スイッチ→抵抗→(GND or CPU) というように電流を受け取る。こうすることでスイッチがオフのときは IN が 0V、オンのときは 5V になる。そのためプルアップでスイッチ操作を行うときの入力される論理値は、スイッチ入力があるときは 1 をスイッチ入力がない時は 0 になる。

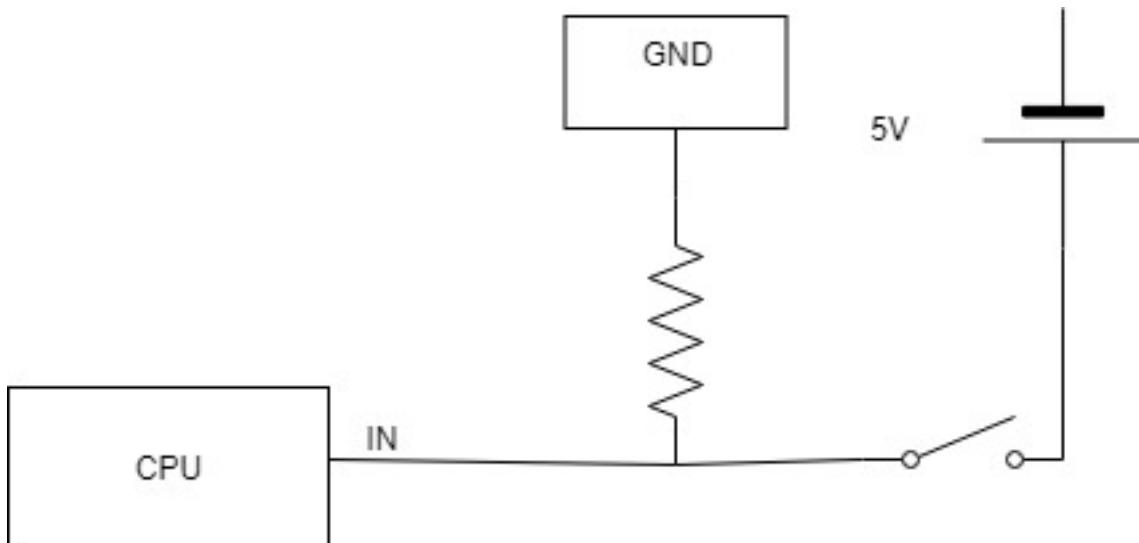


図 12 プルダウン回路図

6 課題 3: Raspberry pi による複数 LED とスイッチの制御

6.1 目的・概要

本実験では、Raspberry pi の電子制御の一つとして複数 LED の点滅とスイッチによる点滅方法の制御プログラムを作成及び実行を行った。具体的には 4 つの LED を順に点滅させ 2 つのスイッチを用いて点滅の停止及び逆順に点滅させる制御を行った。

6.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- Raspberry Pi 3 Model B v1.2 v1.2 (CPU:1.2GHz)
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- HDMI ケーブル
- USB キーボード及びマウス
- Micro USB 電源ケーブル
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- 抵抗 (1k Ω)
- 押しボタンスイッチ
- LED 及び抵抗を実装した基盤

下準備として実験 1,2 同様、Raspberry pi に Micro SD カード、USB キーボード及びマウス、Micro USB 電源ケーブルを接続し、Raspberry pi で電子制御を行えるようにした。また HDMI ケーブルを用いて Raspberry pi とディスプレイを接続し、また、Raspberrypi の GPIO に GPIO ブレッドボード接続ケーブルの T 字型基盤が接続していない側を接続した。T 字型基盤をブレッドボードに接続した。ブレッドボードを使い、図 13 のように回路を作成した。GPIO の 4,17,25,27 にそれぞれ抵抗,LED という順で配線し LED の低電位側を GND に接続した。また、スイッチを GND に接続し、GPIO の 20,21 から抵抗、スイッチ、GND というように電流が流れよう配線した。ソースコード 3 のプログラムコードを実行することで LED の点滅の制御の実行をした。実行コマンドは `python3 sw_led.py`。GPIO の 20 ピンに接続したスイッチを 1、GPIO の 21 ピンに接続したスイッチを 2 とした。プログラムのフローチャートは図 14 である。

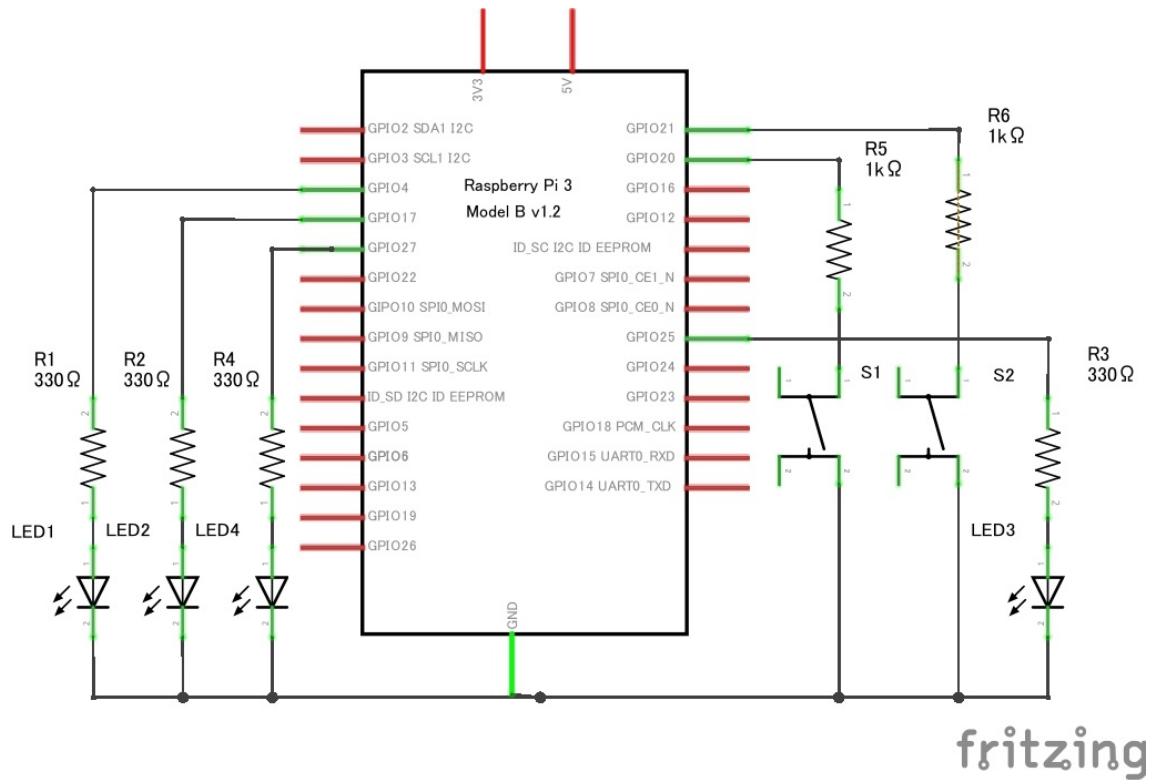


図 13 Rasberry pi と複数 LED 及びスイッチを接続する回路図 R: 抵抗 LED: Light Emitting Diode
GND: グランド S: スイッチ

ソースコード 3: sw_led.py

```

1 import wiringpi as pi, time
2
3 #スイッチ接続 GPIO 端子番号
4 SW_PIN_1 = 20
5 SW_PIN_2 = 21
6 #LED 接続 GPIO 端子番号
7 LED_PIN = [4,17,25,22]
8
9 #スイッチ接続端子を入力モードにする
10 pi.wiringPiSetupGpio()
11 pi.pinMode(SW_PIN_1, pi.INPUT)
12 pi.pinMode(SW_PIN_2, pi.INPUT)
13
14 #LED 接続端子を出力モードにする
15 for i in range(4):
16     pi.pinMode(LED_PIN[i], pi.OUTPUT)
17

```

```

18 #スイッチ接続端子をプルアップモードにする
19 pi.pullUpDnControl(SW_PIN_1, pi.PUD_UP)
20 pi.pullUpDnControl(SW_PIN_2, pi.PUD_UP)

21
22 #前回終了時の LED の状態を確認する
23 flag = 0
24 for i in range(4):
25     if (pi.digitalRead(LED_PIN[i]) == 0):
26         flag = 1
27         break

28
29 #もし光っているものがなければ LED0 接続端子を High にする
30 if not flag:
31     pi.digitalWrite(LED_PIN[0], pi.HIGH)

32
33 #スイッチのオンオフに関する変数
34 key_1 = 1
35 key_2 = 1
36 pre_1 = 1
37 pre_2 = 1

38
39 while True:
40     if (pre_1 == 1):#もし前回ループ終了時にスイッチ 1 が押されていないならば
41         if (pi.digitalRead(SW_PIN_1) == 0): #スイッチ 1 が押されているならば
42             key_1 ^= 1 #スイッチ 1 の状態を切り替える

43
44     if (pre_2 == 1):#もし前回ループ終了時にスイッチ 2 が押されていないならば
45         if (pi.digitalRead(SW_PIN_2) == 0): #スイッチ 2 が押されているならば
46             key_2 ^= 1 #スイッチ 2 の状態を切り替える

47
48     for i in range(4):
49         if (pi.digitalRead(LED_PIN[i]) == 1):
50             if key_1: #もしスイッチ 1 がオフならば
51                 if key_2: #もしスイッチ 2 がオフならば 0,1,2,3,0,1,2,…の順に光らせる
52                     pi.digitalWrite(LED_PIN[(i+1)%4], pi.HIGH)
53                     pi.digitalWrite(LED_PIN[i], pi.LOW)

54
55             else: #もしスイッチ 2 がオンならば 3,2,1,0,3,2,1,…の順に光らせる
56                 pi.digitalWrite(LED_PIN[(i-1)%4], pi.HIGH)
57                 pi.digitalWrite(LED_PIN[i], pi.LOW)

```

```
58     break  
59  
60     #次回ループのためにスイッチの状態を読み込む  
61     pre_1 = pi.digitalRead(SW_PIN_1)  
62     pre_2 = pi.digitalRead(SW_PIN_2)  
63     time.sleep(0.1)
```

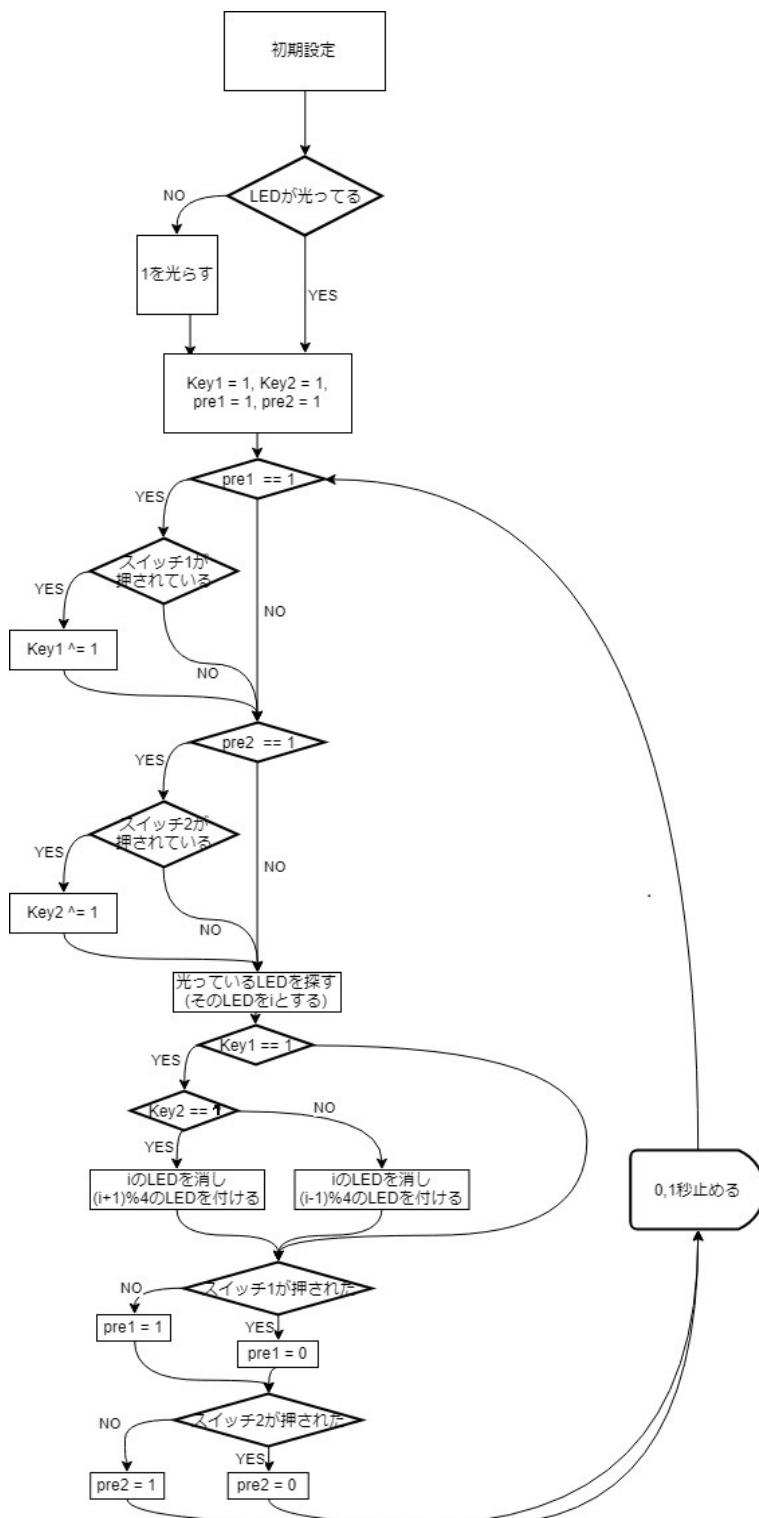


図 14 ソースコード 3 のフローチャート

6.3 実験結果

LED が

○●●●→●○●●→●●○●→●●●○→（最初に戻る）

といった具合に点滅した（○が光っているランプ）。スイッチ 1 を押すことで点滅が停止、スイッチ 2 を押すことで点滅順の逆転が起こった。しかし、LED の明かりが変わった直後にスイッチを押し離したところ、スイッチが押されたことは考慮されなかった。

6.4 考察

明かりが変わった直後に同一スイッチ押し離しても入力を認識しなかった。これはプログラムでスイッチを認識する部分は sw_led.py の 40 行目 46 行目及び 61,62 行目であるが、この部分の問題ではなく、time.sleep(0.1) で 0.1 秒間実行が停止している間にスイッチを押し離しすると、このようにスイッチが認識されないことが起こりうると考えた。そのため下記のソースコード 3-1: new_sw_led.py を python3 new_sw_led.py で実行することで解消すると考えられる。これは元のソースコード 3 の time.sleep(0.1) の 0.1 秒間の LED の変化しない時間を time.sleep(0.01) を 10 回繰り返し、繰り返しの度にスイッチの入力を受け付けるという仕様である。

ソースコード 3-1: new_sw_led.py

```
1 import wiringpi as pi, time
2
3 #スイッチ接続 GPIO 端子番号
4 SW_PIN_1 = 20
5 SW_PIN_2 = 21
6 #LED 接続 GPIO 端子番号
7 LED_PIN = [4,17,25,22]
8
9 #スイッチ接続端子を入力モードにする
10 pi.wiringPiSetupGpio()
11 pi.pinMode(SW_PIN_1, pi.INPUT)
12 pi.pinMode(SW_PIN_2, pi.INPUT)
13
14 #LED 接続端子を出力モードにする
15 for i in range(4):
16     pi.pinMode(LED_PIN[i], pi.OUTPUT)
17
18 #スイッチ接続端子をプルアップモードにする
19 pi.pullUpDnControl(SW_PIN_1, pi.PUD_UP)
20 pi.pullUpDnControl(SW_PIN_2, pi.PUD_UP)
```

```

21
22 #前回終了時の LED の状態を確認する
23 flag = 0
24 for i in range(4):
25     if (pi.digitalRead(LED_PIN[i]) == 0):
26         flag = 1
27         break
28
29 #もし光っているものがなければ LED0 接続端子を High にする
30 if not flag:
31     pi.digitalWrite(LED_PIN[0], pi.HIGH)
32
33 #スイッチのオンオフに関する变数
34 key_1 = 1
35 key_2 = 1
36 pre_1 = 1
37 pre_2 = 1
38
39 while True:
40     if (pre_1 == 1):#もし前回ループ終了時にスイッチ 1 が押されていないならば
41         if (pi.digitalRead(SW_PIN_1) == 0): #スイッチ 1 が押されているならば
42             key_1 ^= 1 #スイッチ 1 の状態を切り替える
43
44     if (pre_2 == 1):#もし前回ループ終了時にスイッチ 2 が押されていないならば
45         if (pi.digitalRead(SW_PIN_2) == 0): #スイッチ 2 が押されているならば
46             key_2 ^= 1 #スイッチ 2 の状態を切り替える
47
48     for i in range(4):
49         if (pi.digitalRead(LED_PIN[i]) == 1):
50             if key_1: #もしスイッチ 1 がオフならば
51                 if key_2: #もしスイッチ 2 がオフならば 0,1,2,3,0,1,2,…の順に光らせる
52                     pi.digitalWrite(LED_PIN[(i+1)%4], pi.HIGH)
53                     pi.digitalWrite(LED_PIN[i], pi.LOW)
54
55             else: #もしスイッチ 2 がオンならば 3,2,1,0,3,2,1,…の順に光らせる
56                 pi.digitalWrite(LED_PIN[(i-1)%4], pi.HIGH)
57                 pi.digitalWrite(LED_PIN[i], pi.LOW)
58             break
59
60 #次回ループのためにスイッチの状態を読み込む

```

```

61     pre_1 = pi.digitalRead(SW_PIN_1)
62     pre_2 = pi.digitalRead(SW_PIN_2)
63
64     #更新点 (0.1秒を 10分割する)
65     for _ in range(10):
66         if (pi.digitalRead(SW_PIN_1) != pre_1):
67             #スイッチが異なる状態になった時その状態を記憶する
68             pre_1 ^= 1
69         if (pi.digitalRead(SW_PIN_1) == 0):
70             #スイッチが異なる状態になった時その状態を記憶する
71             pre_2 ^= 1
72         time.sleep(0.01)

```

7 課題 4: Raspberry pi による単一ステッピングモータの制御

7.1 目的・概要

本実験では、Raspberry pi の電子制御の一つとして単一ステッピングモータの制御プログラムの作成及び実行を行う。具体的には、ステッピングモータを Raspberry pi に接続しユニポーラ駆動の 1 相、2 相、1-2 相励磁の正転逆転停止をそれぞれ実行する。

7.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- Raspberry Pi 3 Model B
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- HDMI ケーブル
- USB キーボード及びマウス
- Micro USB 電源ケーブル
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- ステッピングモータ (28BYJ-48: ギア比 64: ステップ角: 11.25°)
- ステッピングモータ制御回路
- 2.1mm DC ジャック
- USB DC5V to DC12V 昇圧ケーブル
- モバイルバッテリー (ELECOM: 10050mAh モバイルバッテリー)(5V/3.6A)

下準備として実験 1,2,3 同様、Raspberry pi に Micro SD カード、USB キーボード及びマウス、Micro USB 電源ケーブルを接続し、Raspberry pi で電子制御を行えるようにした。また HDMI ケーブルを用いて Raspberry pi とディスプレイを接続し、また、Raspberrypi の GPIO に GPIO ブレッドボード接続ケーブルの T 字型基盤が接続していない側を接続した。T 字型基盤をブレッドボードに接続した。ブレッドボードを使い、図 15 のように回路を作成した。GPIO の 6, 13, 19, 26 をステッピングモータ駆動回路に接続しそれぞれの色に対応するようモータと駆動回路を配線した。また、DC ジャックからステッピングモータ駆動回路に 12V、駆動回路の GND と Raspberry pi の GND に DC ジャックの低電圧部を接続した。昇圧ケーブルに DC ジャックとモバイルバッテリーを接続しブレッドボードにつなげた。実態配線図は図 16 である。ソースコード 4,5,6 のプログラムコードを実行することでステッピングモータの制御の実行をした。実行コマンドは 1 相励磁,2 相励磁,1-2 相励磁それぞれ python3 step1_ras.py [reverse], python3 step2_ras.py [reverse], python3 step1-2_ras.py [reverse], ([reverse] は 0 なら正転、1 なら後転) とした。ステッピングモータが一周するたびに 2 秒停止するプログラムとなっている。

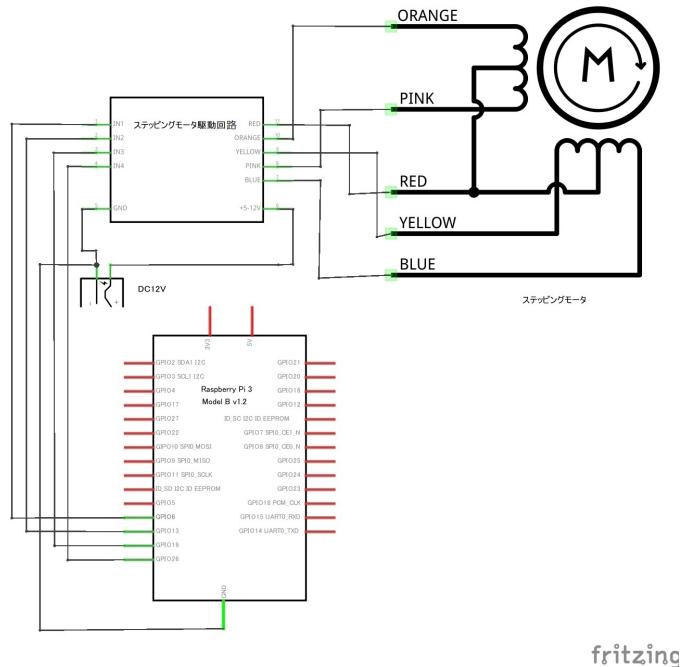


図 15 Rasberry pi と単一ステッピングモータと駆動回路を接続する回路図 Moter: ステッピングモータ GND: グランド Power Plug:

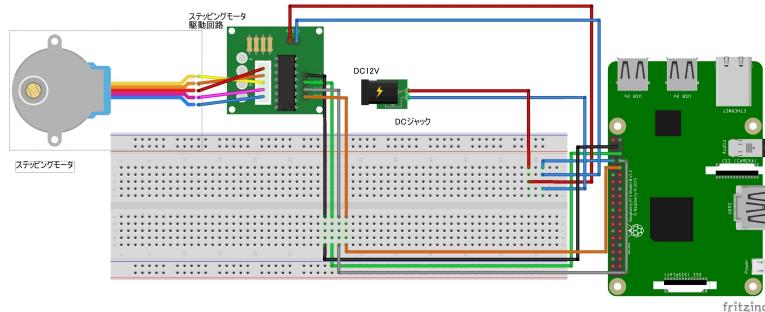


図 16 Raspberry pi と単一ステッピングモータと駆動回路を接続する実態配線図

ソースコード 4: step1_ras.py

```

1 import wiringpi as pi, time
2 import sys
3 reverse = int(sys.argv[1]) #コマンドライン引数により正転逆転に関する変数を受け取る
4 T = 0.002 #周期
5 STEP_PIN = [6,13,19,26] #各相に対する GPIO 端子
6
7 #出力モードにする
8 pi.wiringPiSetupGpio()
9 for i in range(4):
10     pi.pinMode(STEP_PIN[i], pi.OUTPUT)
11
12 #駆動
13 while True:
14     for turn in range(512): #1周したら一旦 2秒停止 (計 2048 回のパルス信号)
15         if reverse == 0: #正転
16             for i in range(4):
17                 #電圧を HIGH にする端子を選ぶ
18                 for j in range(4):
19                     #HIGH にする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
20                     if j == i:
21                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
22                     else:
23                         pi.digitalWrite(STEP_PIN[j], pi.LOW)
24                     time.sleep(T)
25
26         else: #逆転
27             for i in range(4)[::-1]:
28                 #電圧を HIGH にする端子を選ぶ

```

```

29         for j in range(4):
30             #HIGHにする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
31             if j == i:
32                 pi.digitalWrite(STEP_PIN[j], pi.HIGH)
33             else:
34                 pi.digitalWrite(STEP_PIN[j], pi.LOW)
35             time.sleep(T)
36         time.sleep(2)

```

ソースコード 5: step2_ras.py

```

1 import wiringpi as pi, time
2 import sys
3 reverse = int(sys.argv[1]) #コマンドライン引数により正転逆転に関する変数を受け取る
4 T = 0.002 #周期
5 STEP_PIN = [6,13,19,26] #各相に対する GPIO 端子
6
7 #出力モードにする
8 pi.wiringPiSetupGpio()
9 for i in range(4):
10     pi.pinMode(STEP_PIN[i], pi.OUTPUT)
11
12 #駆動
13 while True:
14     for turn in range(512): #1周したら一旦 2秒停止 (2048回のパルス信号)
15         if reverse == 0: #正転
16             for i in range(4):
17                 #電圧を HIGH にする端子を選ぶ
18                 for j in range(4):
19                     #HIGHにする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
20                     #2相励磁なため (i+1)%4 の端子も HIGH にする
21                     if j == i:
22                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
23                     elif j == (i+1)%4:
24                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
25                     else:
26                         pi.digitalWrite(STEP_PIN[j], pi.LOW)
27                     time.sleep(T)
28
29         else: #逆転

```

```

30     for i in range(4)[::-1]:
31         #電圧を HIGH にする端子を選ぶ
32         for j in range(4):
33             #HIGH にする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
34             #2相励磁なため (i+1)%4 の端子も HIGH にする
35             if j == i:
36                 pi.digitalWrite(STEP_PIN[j], pi.HIGH)
37             elif j == (i+1)%4:
38                 pi.digitalWrite(STEP_PIN[j], pi.HIGH)
39             else:
40                 pi.digitalWrite(STEP_PIN[j], pi.LOW)
41             time.sleep(T)
42         time.sleep(2)

```

ソースコード 6: step1-2_ras.py

```

1 import wiringpi as pi, time
2 import sys
3 reverse = int(sys.argv[1]) #コマンドライン引数により正転逆転に関する変数を受け取る
4 T = 0.002 #周期
5 STEP_PIN = [6,13,19,26] #各相に対する GPIO 端子
6
7 #出力モードにする
8 pi.wiringPiSetupGpio()
9 for i in range(4):
10     pi.pinMode(STEP_PIN[i], pi.OUTPUT)
11
12 #駆動
13 while True:
14     for turn in range(512): #1周したら一旦 2秒停止 (4096 回のパルス信号)
15         if reverse == 0: #正転
16             for i in range(8):
17                 #電圧を HIGH にする端子を選ぶ
18                 k = i//2
19                 if i%2 == 0: #偶数回目なら 1 相励磁
20                     for j in range(4):
21                         #HIGH にする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
22                         if j == k:
23                             pi.digitalWrite(STEP_PIN[j], pi.HIGH)
24                         else:

```

```

25         pi.digitalWrite(STEP_PIN[j], pi.LOW)
26     else: #奇数回目なら 2相励磁
27         for j in range(4):
28             #HIGHにする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
29             #2相励磁なため (i+1)%4 の端子も HIGH にする
30             if j == k:
31                 pi.digitalWrite(STEP_PIN[j], pi.HIGH)
32             elif j == (k+1)%4:
33                 pi.digitalWrite(STEP_PIN[j], pi.HIGH)
34             else:
35                 pi.digitalWrite(STEP_PIN[j], pi.LOW)
36             time.sleep(T)
37
38     else: #逆転
39         for i in range(8)[::-1]:
40             #電圧を HIGH にする端子を選ぶ
41             k = i//2
42             if i%2 == 0: #偶数回目なら 1相励磁
43                 for j in range(4):
44                     #HIGHにする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
45                     if j == k:
46                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
47                     else:
48                         pi.digitalWrite(STEP_PIN[j], pi.LOW)
49             else: #奇数回目なら 2相励磁
50                 for j in range(4):
51                     #HIGHにする端子と同じ場合は HIGH そうでない場合は LOW を選ぶ
52                     #2相励磁なため (i+1)%4 の端子も HIGH にする
53                     if j == k:
54                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
55                     elif j == (k+1)%4:
56                         pi.digitalWrite(STEP_PIN[j], pi.HIGH)
57                     else:
58                         pi.digitalWrite(STEP_PIN[j], pi.LOW)
59             time.sleep(T)
60
61     time.sleep(2)

```

7.3 実験結果

ステッピングモータの駆動方式である 1 相励磁、2 相励磁、1-2 相励磁方式のそれぞれについて、ステッピングモータの回転及び逆転、停止をしていた。

7.4 考察

今回の実験でステッピングモータ駆動回路の故障によりステッピングモータが一度駆動不可能な状態に陥った。なぜ駆動回路が故障したのか考察する。まず、ステッピングモータが駆動不能になった際に他で実際に使用可能な配線ケーブルに付け替えたが状態が変わらず、駆動回路を変更したところ駆動を再開したため駆動回路の故障であると判断した。駆動回路にはドライバボード上に駆動用 IC がある。この中にはコンデンサやレジスタが存在し、電子パルスの電圧を制御している。取り外しをした際この部分が非常に熱されていた。そのためドライバボードの駆動用 IC が熱損傷を起こしていた為に故障したと考えられる。熱損傷を防ぐには長時間のステッピングモータの駆動を防ぐ必要があるため、必要のない時はステッピングモータに電源を接続しないことが防止策として考えられる。

8 課題 5: Raspberry pi による複数ステッピングモータの制御

8.1 目的・概要

本実験では Raspberry pi を用いてプログラムを実行しステッピングモータの正転/逆転/停止制御をスイッチを用いて行う。具体的には二つのスイッチを準備し、その信号によって正転/逆転/停止の状態を切り替えるようなプログラムを作成し、2 つのステッピングモータを制御する。

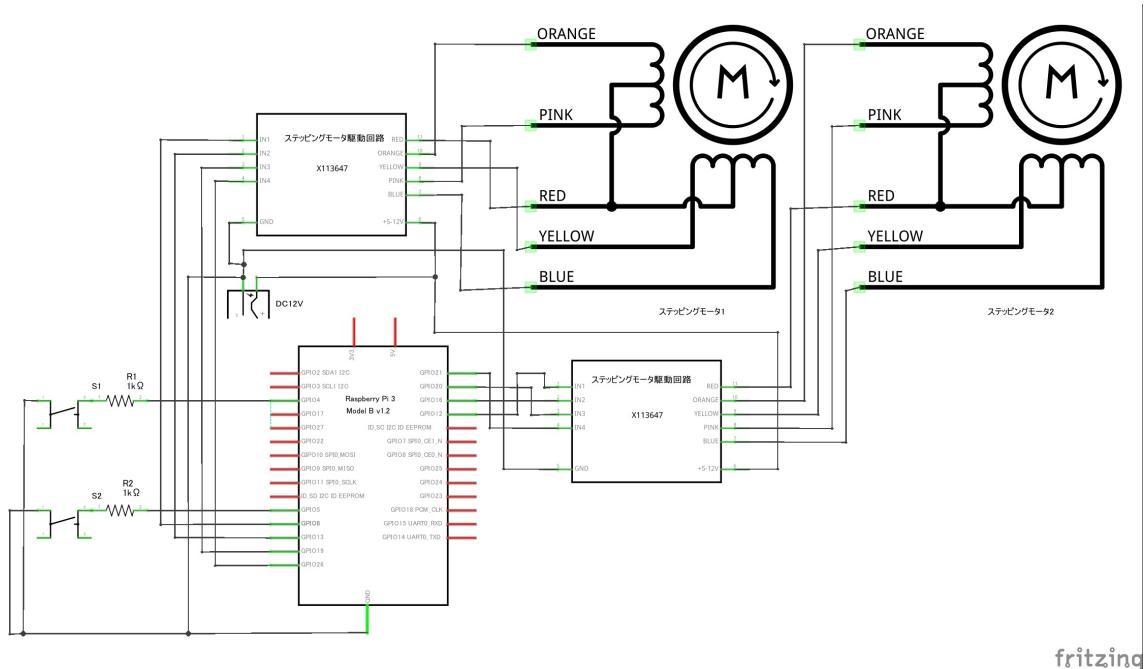
8.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- Raspberry Pi 3 Model B
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- HDMI ケーブル
- USB キーボード及びマウス
- Micro USB 電源ケーブル
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- ステッピングモータ (28BYJ-48 ギア比 64: ステップ角: 11.25°)
- ステッピングモータ制御回路
- 2.1mm DC ジャック

- USB DC5V to DC12V 升圧ケーブル
- モバイルバッテリー (ELECOM: 10050mAh モバイルバッテリー)(5V/3.6A)
- 押しボタンスイッチ
- 抵抗 1k Ω

下準備として実験 1,2,3,4 同様、Raspberry pi に Micro SD カード、USB キーボード及びマウス、Micro USB 電源ケーブルを接続し、Raspberry pi で電子制御を行えるようにした。また HDMI ケーブルを用いて Raspberry pi とディスプレイを接続し、また、Raspberrypi の GPIO に GPIO ブレッドボード接続ケーブルの T 字型基盤が接続していない側を接続した。T 字型基盤をブレッドボードに接続した。ブレッドボードを使い、図 17 のように回路を作成した。GPIO の 6, 13, 19, 26 をステッピングモータ駆動回路 1 に 12,16,20,21 をステッピングモータ駆動回路 2 に接続しそれぞれの色に対応するようモータと駆動回路を配線した。また、DC ジャックから二つのステッピングモータ駆動回路に 12V、駆動回路の GND と Raspberry pi の GND に DC ジャックの低電圧部を接続した。昇圧ケーブルに DC ジャックとモバイルバッテリーを接続しブレッドボードにつなげた。また、GPIO4 と 5 ピンから抵抗、スイッチ、GND の順で配線した。実態配線図は図 18 である。ソースコード 7 のプログラムコードを実行することでスイッチによるステッピングモータの制御の実行をした。実行コマンドは python3 sw_steo.py とした。GPIO の 4 ピンにつながっているスイッチをスイッチ 1、GPIO の 5 ピンにつながっているスイッチをスイッチ 2 とした。スイッチ 1 を押すと正転逆転が変更され、スイッチ 2 を押すとモータが停止する。その後スイッチ 1 を押すとモータの回転が再開される仕様とした。プログラムに前の 2 つのスイッチの状態を保持する変数 pre_1,pre_2 作成することでこの仕様を満たした。プログラムのフローチャートは図 19 である。



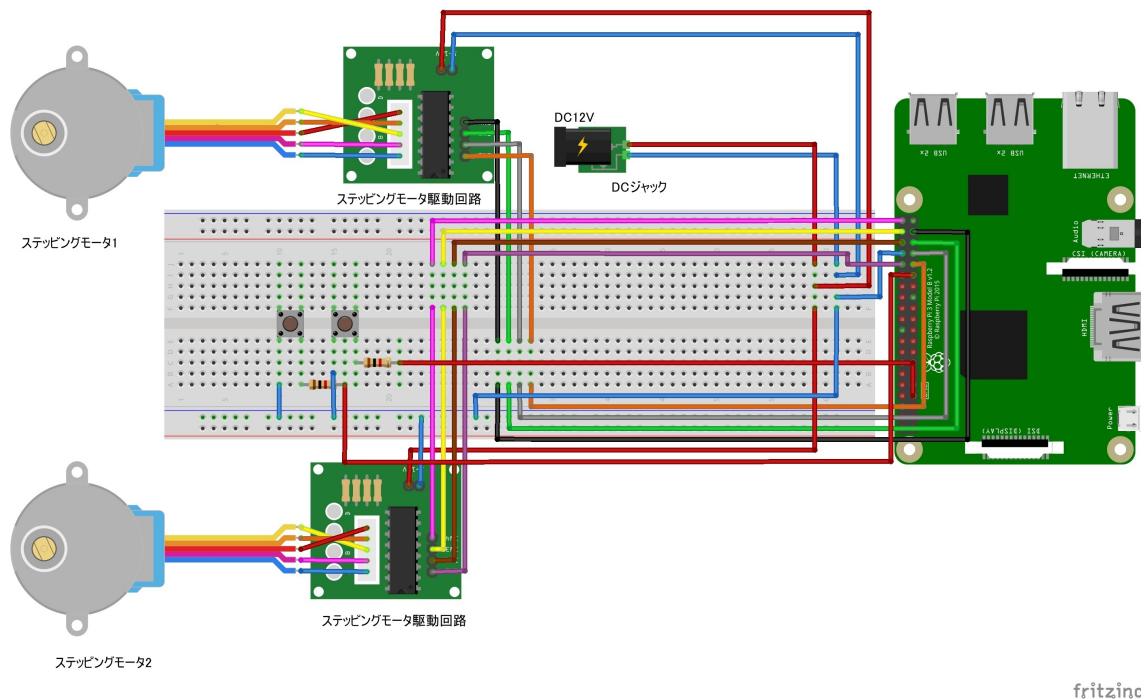


図 18 Raspberry pi と複数ステッピングモータ及びスイッチを接続する実態配線図

ソースコード 7: sw_step.py

```

1 import wiringpi as pi, time
2
3 T = 0.002 #周期
4 STEP1_PIN = [6,13,19,26] #ステッピングモータ 1 の各相に対する GPIO 端子
5 STEP2_PIN = [12,16,20,21] #ステッピングモータ 2 の各相に対する GPIO 端子
6
7 #ステッピングモータ接続端子を出力モードにする
8 pi.wiringPiSetupGpio()
9 for i in range(4):
10     pi.pinMode(STEP1_PIN[i], pi.OUTPUT)
11     pi.pinMode(STEP2_PIN[i], pi.OUTPUT)
12
13 #スイッチ接続 GPIO 端子番号
14 SW1_PIN = 4
15 SW2_PIN = 5
16
17 #スイッチ接続端子を入力モードにする
18 pi.wiringPiSetupGpio()
19 pi.pinMode(SW1_PIN, pi.INPUT)

```

```

20 pi.pinMode(SW2_PIN, pi.INPUT)
21
22 #スイッチ接続端子をプルアップモードにする
23 pi.pullUpDnControl(SW1_PIN, pi.PUD_UP)
24 pi.pullUpDnControl(SW2_PIN, pi.PUD_UP)
25
26 key_1 = 1 #スイッチ1オンオフに関する変数
27 key_2 = 1 #スイッチ2オンオフに関する変数
28 pre_1 = 1 #スイッチ1の前回状態保存変数
29 pre_2 = 1 #スイッチ2の前回状態保存変数
30
31 while True:
32     print(key_1,key_2,pre_1,pre_2) #スイッチの状態をモニタに表示
33     if (pre_1 == 1):
34         if (pi.digitalRead(SW1_PIN) == 0):
35             key_1 ^= 1
36             pre_1 = 0
37     if (pre_2 == 1):
38         if (pi.digitalRead(SW2_PIN) == 0):
39             key_2 ^= 1
40             pre_2 = 0
41     if not key_1: #もしスイッチ1がオンなら
42         if key_2: #もしスイッチ2がオフなら
43             for i in range(4): #2つとも正転
44                 if (pre_1 == 1):
45                     if (pi.digitalRead(SW1_PIN) == 0):
46                         key_1 ^= 1
47                 if (pre_2 == 1):
48                     if (pi.digitalRead(SW2_PIN) == 0):
49                         key_2 ^= 1
50             for j in range(4):
51                 if j == i:
52                     pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
53                     pi.digitalWrite(STEP2_PIN[j], pi.HIGH)
54             else:
55                 pi.digitalWrite(STEP1_PIN[j], pi.LOW)
56                 pi.digitalWrite(STEP2_PIN[j], pi.LOW)
57     pre_1 = pi.digitalRead(SW1_PIN)
58     pre_2 = pi.digitalRead(SW2_PIN)
59     time.sleep(T)

```

```

60
61     else: #もしスイッチ 2がオンなら
62         for i in range(4): #ステッピングモータ 1は正転、ステッピングモータ 2は逆転
63             if (pre_1 == 1):
64                 if (pi.digitalRead(SW1_PIN) == 0):
65                     key_1 ^= 1
66             if (pre_2 == 1):
67                 if (pi.digitalRead(SW2_PIN) == 0):
68                     key_2 ^= 1
69             for j in range(4):
70                 if j == i:
71                     pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
72                     pi.digitalWrite(STEP2_PIN[3-j], pi.HIGH)
73                 else:
74                     pi.digitalWrite(STEP1_PIN[j], pi.LOW)
75                     pi.digitalWrite(STEP2_PIN[3-j], pi.LOW)
76             pre_1 = pi.digitalRead(SW1_PIN)
77             pre_2 = pi.digitalRead(SW2_PIN)
78             time.sleep(T)
79
80     else: #もしスイッチ 1がオフなら
81         for i in range(4):
82             pi.digitalWrite(STEP1_PIN[i], pi.LOW)
83             pi.digitalWrite(STEP2_PIN[i], pi.LOW)
84             pre_1 = pi.digitalRead(SW1_PIN)
85             pre_2 = pi.digitalRead(SW2_PIN)

```

8.3 実験結果

ステッピングモータについて、スイッチ 1 を押すたびにステッピングモータの回転が逆転し、スイッチ 2 を押すと回転が停止し再び 2 を押すと回転を始めた。スイッチ 2 で停止をしている時にスイッチ 1 を押しスイッチ 2 を押すと停止する前と反対方向に回り始めた。

8.4 考察

今回の実験のプログラムではスイッチの読み込みは 0.002 秒ごとに発生させ、実行に影響させる最大時間は 0.008 秒である。チャタリング発生時間 100 秒 0.01 秒であるためチャタリング発生が起こる可能性があるためチャタリングを防ぐには T の値を 0.003 秒程度まで上げる必要があると考えられる。

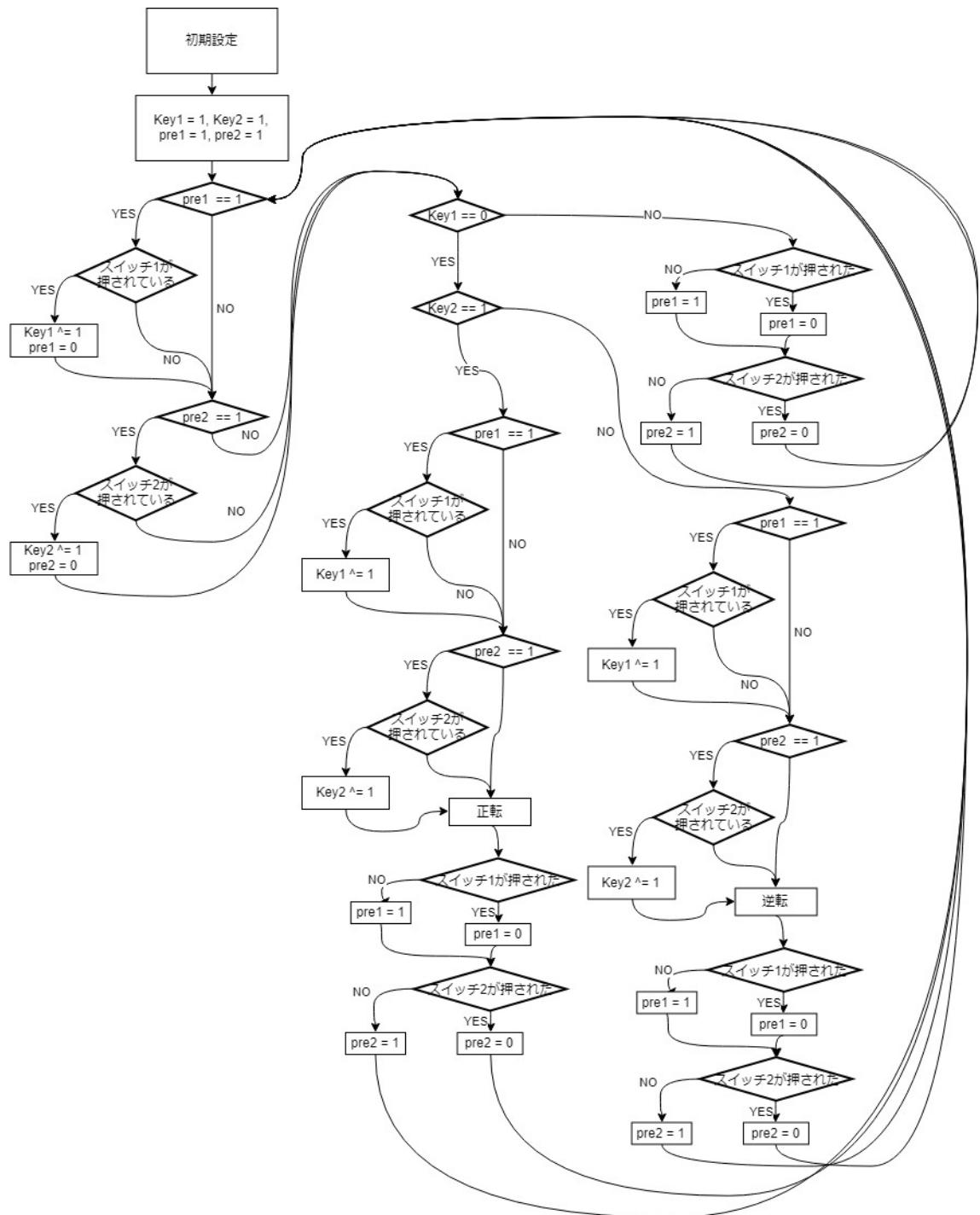


図 19 ソースコード 7 のフローチャート

9 課題 6: マニュアル操作ステッピングモータカーの作成

9.1 目的・概要

本実験では Raspberry pi を用いてマニュアル操作ステッピングモータカーを作成し制御を行う。具体的には Raspberry pi とステッピングモータ、スイッチをつなげる回路を作成し、LEGO ブロックを組み立てステッピングモータカーとし、これに作成した回路を乗せマニュアル操作可能なステッピングモータカーを作成し、前進、後進、右折、左折、停止させる。

9.2 実験方法

はじめに、今回の実験を行うための実験道具の構成は以下である。

- PC/AT 互換機
- Raspberry Pi 3 Model B
- Micro SD カード (NOOBS (OS インストーラ) 書き込み済み)
- USB キーボード及びマウス
- Micro USB 電源ケーブル (USB 接続)
- ディスプレイ (23 型 IPS 方式三菱液晶ディスプレイ (ノングレア) RDT232WX(BK)(メーカー:三菱電機株式会社)(シリアルナンバー:11230667AJ)
- ブレッドボード
- T 字型基盤 (GPIO ブレッドボード接続ケーブル接続済み)
- 配線ケーブル
- ステッピングモータ (28BYJ-48 ギア比 64: ステップ角: 11.25°)
- ステッピングモータ制御回路
- 2.1mm DC ジャック
- USB DC5V to DC12V 昇圧ケーブル
- モバイルバッテリー (ELECOM: 10050mAh モバイルバッテリー)(5V/3.6A)
- 押しボタンスイッチ
- 抵抗 1k Ω
- LEGO ブロック
- ステッピングモータマウンタ
- ガムテープ

まず初めに、Raspberry pi の遠隔操作を可能にするために SSH サーバを有効にし、Bonjour 用ホスト名の設定を Raspberry pi で行った。同一ネットワーク内に存在する AT/PC 互換機から Raspberry pi へ SSH 接続をした。次にレゴブロックを組み立てステッピングモータの外形を作成した。USB キーボード及びマウス、ディスプレイを PC/AT 互換機に接続し PC/AT 互換機の操作を可能とした。Raspberry pi に Micro SD カードモバイルバッテリーに接続した Micro USB 電源ケーブルを接続し Raspberry pi を操作可能にした。Raspberry pi、スイッチ、抵抗、ステッピングモータ及び駆動回路、モバイルバッテリーを図 20 のような回路で接続し、これをステッピングモータカーにのせた。GPIO の 6、13、19、26 をステッピングモータ

駆動回路 1 に 12,16,20,21 をステッピングモータ駆動回路 2 に 18、23、24、25 をステッピングモータ駆動回路 3 に接続しそれぞれの色に対応するようモータと駆動回路を配線した。また、DC ジャックからすべてのステッピングモータ駆動回路に 12V、駆動回路の GND と Raspberry pi の GND に DC ジャックの低電圧部を接続した。昇圧ケーブルに DC ジャックとモバイルバッテリーを接続しブレッドボードにつなげた。また、GPIO4 と 5 と 17 と 22 ピンから抵抗、スイッチ、GND の順で配線した。21 である。作成したステッピングモータカーは図 22 のとおりである。次にソースコード 8 のプログラムを Raspberry pi で作成し PC/AT 互換機から遠隔で実行をし、ステッピングモータカーのマニュアル操作制御を行った。実行コマンドは python3 drive.py で行った。プログラムのフローチャートは図 23 のとおり。

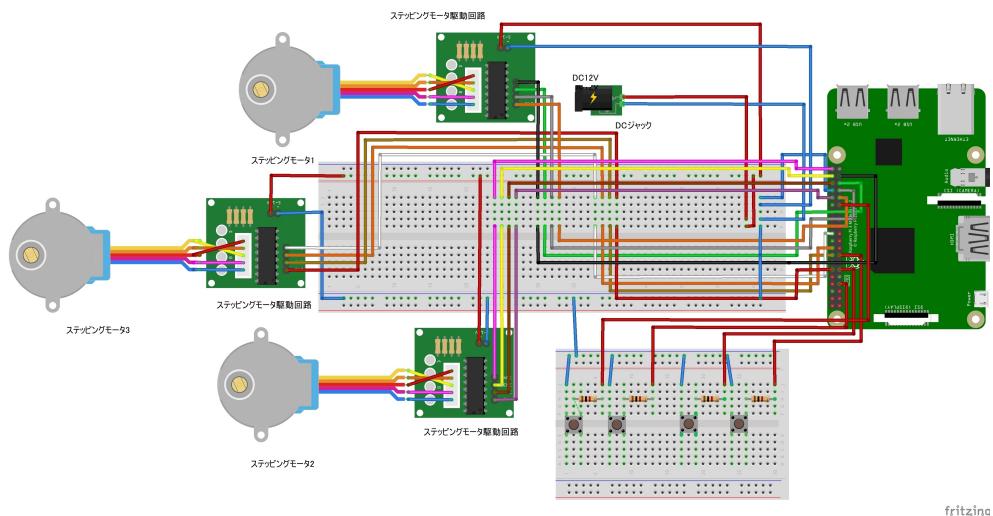


図 20 Raspberry pi と複数ステッピングモータ及びスイッチを接続したステッピングモータカーの回路図
R: 抵抗 Motor: ステッピングモータ GND: グランド S: スイッチ

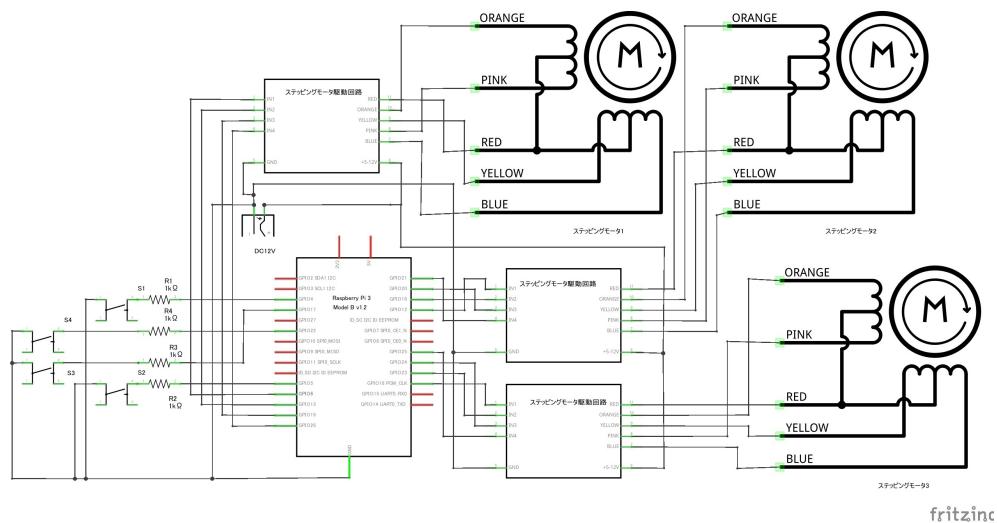


図 21 Raspberry pi と複数ステッピングモータ及びスイッチを接続したステッピングモータカーの実態配線図

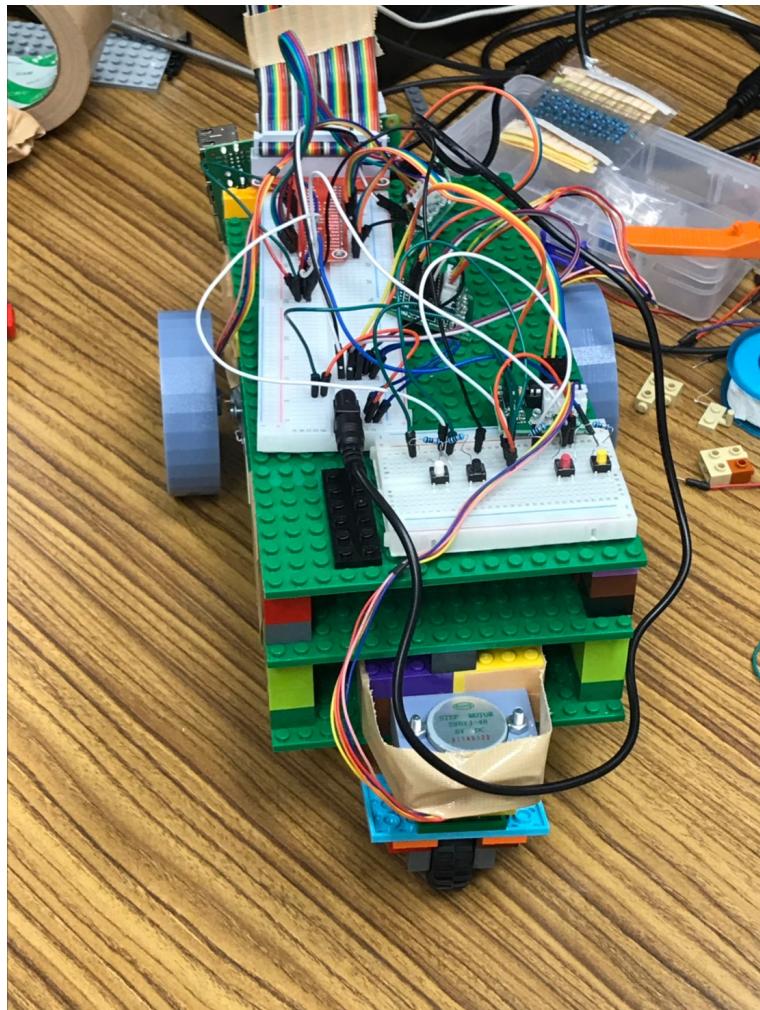


図 22 作成したステッピングモータカー

ソースコード 8: drive.py

```
1 import wiringpi as pi, time
2
3 def rotate(reverse, t): #回転用関数
4     if reverse == 0:
5         for i in range(4): #正転
6             for j in range(4):
7                 if j == i or j == (i+1)%4:
8                     pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
9                     pi.digitalWrite(STEP2_PIN[3-j], pi.HIGH)
10            else:
11                pi.digitalWrite(STEP1_PIN[j], pi.LOW)
12                pi.digitalWrite(STEP2_PIN[3-j], pi.LOW)
```

```

13     time.sleep(t)
14
15     else:
16         for i in range(4)[::-1]: #逆転
17             for j in range(4):
18                 if j == i or j == (i+1)%4:
19                     pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
20                     pi.digitalWrite(STEP2_PIN[3-j], pi.HIGH)
21             else:
22                 pi.digitalWrite(STEP1_PIN[j], pi.LOW)
23                 pi.digitalWrite(STEP2_PIN[3-j], pi.LOW)
24             time.sleep(t)
25
26 T = 0.002 #周期
27 STEP1_PIN = [6,13,19,26] #ステッピングモータ 1 の各相に対する GPIO 端子
28 STEP2_PIN = [12,16,20,21] #ステッピングモータ 2 の各相に対する GPIO 端子
29 STEP3_PIN = [18,23,24,25] #ステッピングモータ 3 の各相に対する GPIO 端子
30
31 #ステッピングモータ接続端子を出力モードにする
32 pi.wiringPiSetupGpio()
33 for i in range(4):
34     pi.pinMode(STEP1_PIN[i], pi.OUTPUT)
35     pi.pinMode(STEP2_PIN[i], pi.OUTPUT)
36     pi.pinMode(STEP3_PIN[i], pi.OUTPUT)
37
38 #スイッチ接続 GPIO 端子番号
39 SW_PIN = [4,5,17,22]
40
41 #スイッチ接続端子を入力モードにする
42 for i in range(4):
43     pi.pinMode(SW_PIN[i], pi.INPUT)
44
45 #スイッチ接続端子をプルアップモードにする
46 for i in range(4):
47     pi.pullUpDnControl(SW_PIN[i], pi.PUD_UP)
48
49 key = [1,1] #スイッチオンオフに関する変数
50
51 while True:
52     print(key) #スイッチの状態をモニタに表示

```

```

53     first = True #スイッチの押し始めに関する変数
54     while (pi.digitalRead(SW_PIN[0]) == 0): #スイッチ 1がオンの間
55         if first: #もし押し始めたら
56             key[0] ^= 1
57             first = False
58         if not key[0]: #もしスイッチ 1がオンなら
59             if key[1]: #もしスイッチ 2がオフなら
60                 rotate(0,T)
61             else: #もしスイッチ 2がオンなら
62                 rotate(1,T)
63         else: #もしスイッチ 1がオフなら
64             for i in range(4):
65                 pi.digitalWrite(STEP1_PIN[i], pi.LOW)
66                 pi.digitalWrite(STEP2_PIN[i], pi.LOW)
67                 pi.digitalWrite(STEP3_PIN[i], pi.LOW)
68
69     first = True #スイッチの押し始めに関する変数
70     while (pi.digitalRead(SW_PIN[1]) == 0): #スイッチ 2がオンの間
71         if first: #もし押し始めたら
72             key[1] ^= 1
73             first = False
74         if not key[0]: #もしスイッチ 1がオンなら
75             if key[1]: #もしスイッチ 2がオフなら
76                 rotate(0,T)
77             else: #もしスイッチ 2がオンなら
78                 rotate(1,T)
79         else: #もしスイッチ 1がオフなら
80             for i in range(4):
81                 pi.digitalWrite(STEP1_PIN[i], pi.LOW)
82                 pi.digitalWrite(STEP2_PIN[i], pi.LOW)
83                 pi.digitalWrite(STEP3_PIN[i], pi.LOW)
84
85     while (pi.digitalRead(SW_PIN[2]) == 0): #スイッチ 3がオンの間
86         for i in range(4): #正転(右回転)
87             for j in range(4):
88                 if j == i or j == (i+1)%4:
89                     pi.digitalWrite(STEP3_PIN[j], pi.HIGH)
90                 else:
91                     pi.digitalWrite(STEP3_PIN[j], pi.LOW)
92             time.sleep(T)

```

```
93
94     while (pi.digitalRead(SW_PIN[3]) == 0): #スイッチ4がオンの間
95         for i in range(4)[::-1]: #逆転(左回転)
96             for j in range(4):
97                 if j == i or j == (i+1)%4:
98                     pi.digitalWrite(STEP3_PIN[j], pi.HIGH)
99                 else:
100                     pi.digitalWrite(STEP3_PIN[j], pi.LOW)
101             time.sleep(T)
102
103         if not key[0]: #もしスイッチ1がオンなら
104             if key[1]: #もしスイッチ2がオフなら
105                 rotate(0,T)
106
107             else: #もしスイッチ2がオンなら
108                 rotate(1,T)
109
110         else: #もしスイッチ1がオフなら
111             for i in range(4): #停止
112                 pi.digitalWrite(STEP1_PIN[i], pi.LOW)
113                 pi.digitalWrite(STEP2_PIN[i], pi.LOW)
114                 pi.digitalWrite(STEP3_PIN[i], pi.LOW)
```

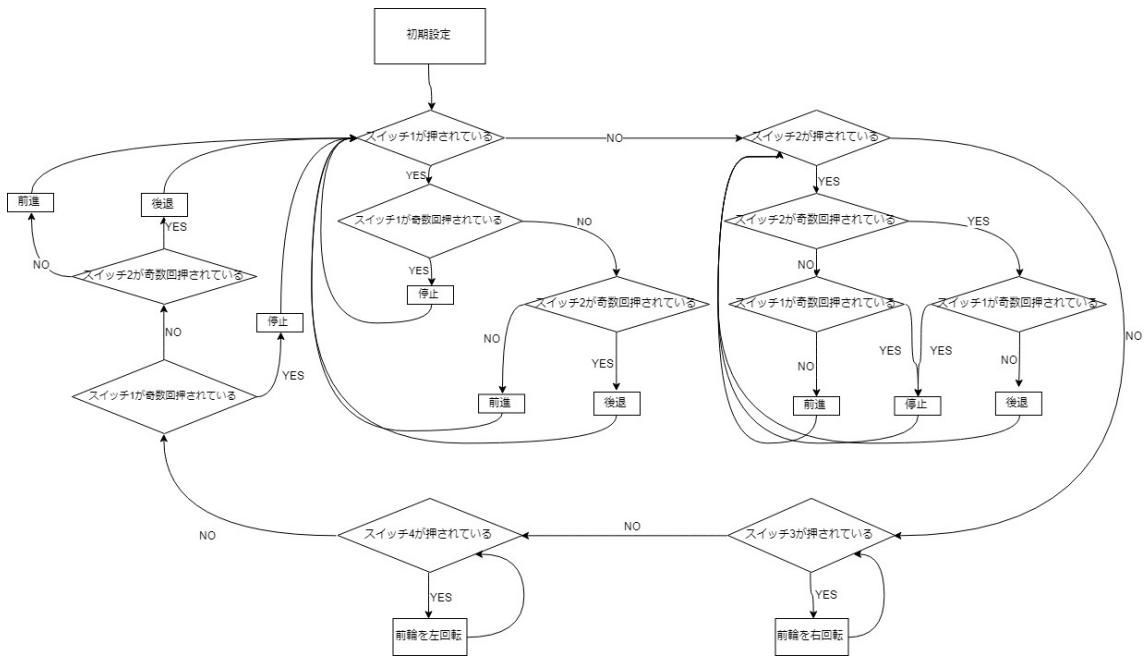


図 23 ソースコード 8 のフローチャート

9.2.1 作成したステッピングモータの仕様

作成したステッピングモータカーは全部で3輪のタイヤを有し、すべてのタイヤがステッピングモータによる駆動制御を受けることができ何らかの移動操作にかかるようになっている。前輪は1つだけで、前方に突出した形で出ておりこちらで、ステッピングモータカーの向きの調節を行う。後輪は2つあり左右についており、ステッピングモータカーの走行に用いる。ステッピングモータカーの後輪は左右同等の回転数で同方向にのみ回転し、その場での回転はできない。そのため旋回は車のような駆動をするため、運転手の運転技術が必要とされる。

スイッチで制御する動作は前進、後退、停止、右折、左折の5種類であるが、すべての操作を別々のスイッチでの制御で行うより、前進及び後退制御を一つのボタンで行う方が良いと判断したためスイッチを4つ行った。4,5,17,22のピンに接続したスイッチをそれぞれ、スイッチ1,2,3,4とする。スイッチ1でスイッチ押すたび停止と駆動を切り替え、スイッチ2でスイッチを押すたびに前進と後退を切り替え、スイッチ3を押している間前輪のタイヤを右向きに回転させ、スイッチ4を押している間前輪のタイヤを左向きに回転させる。前輪のタイヤの向きによりステッピングモータカーの進路を決定する進行方向決定用車輪の役割を果たしている。スイッチ1を押した後停止している間に2のスイッチを奇数回押した後にスイッチ1を再び押すと前進していたなら後退を、後退していたなら前進を行う。又、ステッピングモータカーの車体大きさと、前輪の飛び出し具合から前輪が進行方向に対して90°以上の回転をするとタイヤが外れてしまうため、それ以上の角度の回転は行わないよう注意が必要である。

9.3 実験結果

4つのスイッチを押すと仕様通りに動いていることを確認した。

9.4 考察

本実験で作成したステッピングモータカーの前輪の駆動角度について考える。前輪の回転にかかるプログラムは drive.py の 85 行目から 101 行目にある。本実験に使用されたステッピングモーターのステッピング角は 11.25° であり、ギア比が 64 であるため、プログラムを考えると while 文のループ一回につき 4 回のパルス信号を送るため 1 ループで回転する角度は

$$\frac{11.25 \times 4}{64} = 0.703125^\circ$$

である。この角度の整数倍で前輪は回転をする。車のハンドルのような任意の角度に前輪を回転させることはできないが、実際に車の操作を行うという点においては十分に問題ない角度であると考えられる。また、1 ループは $4 \times T$ 秒以上かかるためおよそ 0.008 秒かかる。そのため、前輪は 0.008 秒ごとに 0.703125° の回転をおこなっていると考えられる。

10 調査課題 6-1 Zeroconf の代表的なプロトコルについて

Zeroconf とはゼロ・コンフィギュレーション・ネットワーキングの略で人手による操作を介さず、又特別なコンフィギュレーションサーバを使わずに、Internet Protocol ネットワークを自動的に作成する技法の事である。通常であればネットワーク管理者が DHCP や DNS サービスの設定をする必要があり、時間がかかり面倒である。しかし Zeroconf により容易にプリンタやコンピュータを自動的にネットワークへ接続することが可能にすることができる。Zeroconf では特に

- ip アドレスの割り当て
- 名前解決
- デバイスの位置の特定

ができるものが関係性がある。ip アドレスの割り当てには APIPA(Automatic Private IP Addressing) というコンピュータやネットワーク機器が自身の IP アドレスを自動的に設定する機能が使われることが多い。

Zeroconf の代表的なプロトコルには以下のものがある。

- マルチキャスト DNS
- DNS-SD
- UPnP SSDP

マルチキャスト DNS と DNS-SD はアップルの方式であり、UPnP SSDP はマイクロソフトの方式である。

通常の DNS はネットワークサーバーにそれぞれのデバイスの IP アドレスを保持し IP アドレスを所得したい場合はサーバーに問い合わせることで名前解決を行う。これをマルチキャスト転送でネットワーク内のデバイスに問い合わせることで名前解決を行うのがマルチキャスト DNS である。しかし、実装によってはマルチキャスト DNS サーバーがローカルリンクネットワーク外の WAN などから送られてきたユニキャストのクエリに応答する場合がある。また、応答パケットのサイズがクエリパケットのサイズより大きく、他のネットワークに対する DOS 攻撃に使用される可能性もある。avahi が Linux での実装。Bounjour がアップルでの実装で使われている。

デバイスの位置特定に使われるのが DNS-SD で DNS Service Discovery という名の通り、ネットワーク上から目的のサービスを提供するノードを探索す機能を有す。Multicast DNS の「DNS resource records」に各ノードのサービス情報を登録し、Well-Known Port Numbers のキーワードによって DNS Query を行い、任意のサービスを提供しているノードの検索をすることができるようとするプロトコルである。後述の SSDP では HTTP メッセージを使用しているのに対してこちらは DNS メッセージを使用している。現在では、Safari ブラウザなどの macOS の多くのネットワーククライアントは手近なサーバの特定に DNS-SD を使っている。Windows 上では、一部のインスタントメッセンジャーや VoIP で DNS-SD をサポートしている。また、Unix 系や Linux ディストリビューションも DNS-SD 機能を備えたものがある。

UPnP は Universal Plug and Play という名前であり、機器を接続するだけでコンピュータネットワークに参加することを可能にするプロトコルである。UPnP は様々なプロトコルに分かれており、ネットワーク上のデバイスの検出を行なうのが UPnP SSDP というプロトコルである。こちらでは HTTP メッセージを使用している。

11 まとめ

本レポートではマイコンの一つである Raspberrypi を用いて電子制御の実験を行った。LED、スイッチ、ステッピングモータの電子制御をプログラムを用いて制御可能にすることを理解した。それらを総合してマニュアル操作のステッピングモータカーを作成した。Raspberry pi の GPIO の通常ピンの使用法について理解した。また、プログラムの実装においてプログラムの停止関数が実際のアナログ操作に及ぼす影響が大きいことを理解した。

参考文献

- [1] Raspberry Pi とは？ | フリエン — (2019/4/30 閲覧)
<https://furien.jp/columns/58/>
- [2] 長いブレッドボード — SWITCHSCIENCE — (2019/4/30 閲覧)
<https://www.switch-science.com/catalog/2293/>
- [3] チャタリングの概要 — マルツオンライン — (2019/5/6 閲覧)
https://www.marutsu.co.jp/pc/static/large_order/1405_311_ph