自動走行多輪駆動車の作成

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 / 5 / 10

目次

1　はじめに

本レポートでは、距離センサ計を用いて，指定コースを自動で走破可能なステッピングモータカーを作成する。駆動距離センサには赤外線式距離センサ計 (Sharp GP2YOA21) を用いる。また、センサから出力されるアナログ信号をデジタル信号へ変換する A/D コンバータとして、A/D コンバータ (MCP3424 with address switch)(DFRobot 製 SKU: DFR0316) を使用する。本レポートでは、自動走行車を作成したのち指定コースを走行させる。その走行から、改善点を考察したのち、本実験のまとめを行う。

1.1　距離センサ計

一般に距離センサ計には超音波による計測法と赤外線による計測法などがある。超音波による距離測定は、非常に短い超音波を前方に発し、反射音が返ってくるまでの時間を計測することで前方までの距離を計測する。これは、音速は常温で、約 340 メートル毎秒で一定なため、音が返るまでの時間が分かれば距離が求められるためで、例えば発射から 10 ミリ秒後に反射音が検出された場合、センサとの距離は約 1.7 メートルとなる。しかし、超音波による距離計測は、その原理上外部の空気の状態により誤差が生じる。また、表面が凸凹している物体や衣服、毛・布素材といった音波を吸収する素材の物体との距離を計測する場合反射音がうまく取れなくなる場合があるため、測定可能性距離が変動するため注意が必要である。赤外線による距離計測は赤外線を前方に発射しすることで距離を計測する。計測する方法として、赤外光の発行側と受講側を同一センサ内でもち三角測量の容量で距離を検出する PSD 方式がある。これは、発光側から出た光が対象物に当たり戻ってきた反射光の入射角により出力電圧を変化させ、その電圧により距離を計測する [1]。本実験の距離センサ計に用いた赤外線センサ計 (Sharp GP2YOA21) はこの PSD 方式で距離を計測する。このセンサは 10cm-80cm までの距離を計測することができ、距離に応じた電圧を約 0.5-3.2V で出力する。出力された電圧は A/D コンバータにてデジタル信号に変換されることで、プログラム内から利用することができる。

1.2　A/D コンバータ

A/D コンバータは受け取ったアナログ値をデジタル値に変換して出力する回路を指す。例えば、本実験のように、距離センサ計から出力される電圧はアナログ値であるため、パソコン等でこの電圧を受け取るためにはこのアナログ値をデジタル値に変換する必要がある。そのため A/D コンバータを使用することで距離センサ計から電圧の入力を受け取ることができるようになる。本実験で使用した A/D コンバータは MCP3424 であり、これを搭載した DFRobot 製の SKU: DFR0316 を実験で用いた。MCP3424 は最大 18bit の分解能をもち、4 チャンネルのコンバータである。制御は I2C で行う。MCP3424 には増幅器が内蔵されており、プログラムから電圧の増幅値を指定することができる。入力をパソコン又はマイコン等で受け取る際に MCP3424 から出力される電圧は 12bit で一秒間に 240 程度のサンプルまで、18bit では 4 程度のサンプルであるためプログラムで受け取る頻度を考慮する必要があり注意が必要である。bit 毎の 1 秒間に出力するサンプルは表 1 である。また MCP3424 の計測可能範囲は ± 2.048V であり，実験では途中に抵抗を挟み降圧することで計測を行う。

bit	Min	Typ(代表値)	Min
12	176	240	328
14	44	60	82
16	11	15	20.5
18	2.75	3.75	5.1

表 1　MCP3424 の DataRate(単位は SPS)

2　実験: 多輪駆動車の自動走行

2.1　概要

A/D コンバータの仕組みを理解を理解したうえで、距離センサ計を用いて，指定コースを自動で走破可能なステッピングモータカーを作成し、指定コースを走行させる。

2.2　実験方法

2.2.1　実験に使用した器具

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 (SSH 接続済)
- 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)
- 抵抗 1k Ω 及び 1.2M Ω
- LEGO ブロック
- ステッピングモータマウンタ
- ガムテープ
- 赤外線式距離センサ計 (Sharp GP2YOA21)
- A/D コンバータ (MCP3424 with address switch)

2.2.2　実験に使用した器具の設定

Raspberry pi は 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操作可能にしたものを使用した。また、同一ネットワーク内に存在する AT/PC 互換機から SSH 接続をした。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。

以下の手順により I2C を有効にした

```
sudo raspi-config
> 5 Interfacing Options
>> P5 I2C
>>> YES
> Finish

sudo reboot
```

その後端末に i2cdetect -y 1　のコマンドを打ち込み MCP3424 のアドレスを確認した。

次に、今後の使用のために赤外線センサ計がどの程度の距離でどの程度の v の値を出力するかを知る必要があったため図 1 の回路を作成し、ソースコード 1 のプログラムを実行することで実験的に確認した。実行コマンドは ad_convert_led_test.py　で行った。このプログラムでは 0.7V 1v が出力されるとき LED のランプが光り、それ以外のときは光らない仕様になっている。図 1 の通り、回路は、Raspberry pi の GPIO 端子の 4 番ピンに 330 Ωの抵抗 LED、GND となるように接続し、GND には A/D コンバータの GND と C1-と赤外線センサ計の GND を接続し、Raspberry pi の 5V を赤外線センサ計の Vcc と A/D コンバータの +5V に接続した。赤外線センサ計の V0 から 1.2M Ωの抵抗 C1+ という順番になるように配線し、A/D コンバータの SDA と SCL を Raspberry pi の SDA と SCL に接続した。

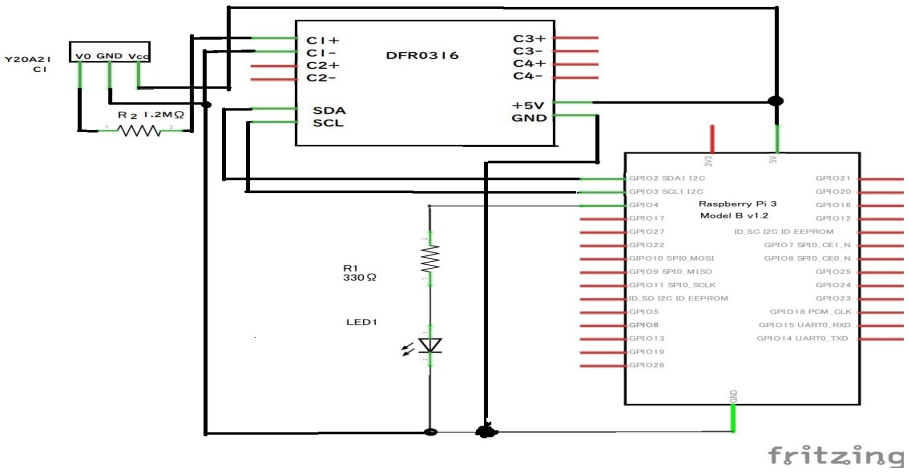

図 1　赤外線センサ計の距離計測用の回路図

ソースコード 1: ad_convert_led_test.py

```
import smbus
import time
import mcp3424
import wiringpi as pi, time

i2c = smbus.SMBus(1)
addr = 0x6e # デバイスアドレス

#LED 接続 GPIO 端子番号
LED_PIN = 4

#LED 接続端子を出力モードにする
pi.wiringPiSetupGpio()
pi.pinMode(LED_PIN, pi.OUTPUT)

while True:
    config = mcp3424.cfg_read | mcp3424.cfg_ch1 | mcp3424.cfg_once |
    ↪ mcp3424.cfg_12bit | mcp3424.cfg_PGAx1
    data = i2c.read_i2c_block_data(addr, config, 2)
    v = mcp3424.to_volt(data, 12)
    print(v)
    if v > 1:#もし電圧が 1 より大きければ点灯
        pi.digitalWrite(LED_PIN, pi.HIGH)

    elif v < 0.7:#もし電圧が 0.7 より小さければ消灯
        pi.digitalWrite(LED_PIN, pi.LOW)
    time.sleep(0.005)
```

2.2.3　走行に用いた回路図及びプログラム

本実験では、Raspberry pi、スイッチ、抵抗、赤外線式距離センサ計、A/D コンバータ及び駆動回路、モバイルバッテリーを図 2 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 を接続し、Raspberry pi の GND に駆動回路の GND と DC ジャックの低電圧部、A/D コンバータの GND、赤外線式距離センサの GND、CH1-、CH2-、CH3-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。Raspberry pi の 5v を A/D コンバータの +5v と赤外線式距離センサの Vcc に接続した。3 つの赤外線式距離センサの V0 から 1.2M Ωの抵抗を接続しそれを、CH1+、2+、3+ にそれぞれ接続した。A/D コンバータ SDA と SCL を Raspberry pi の SDA と SCL に接続した。

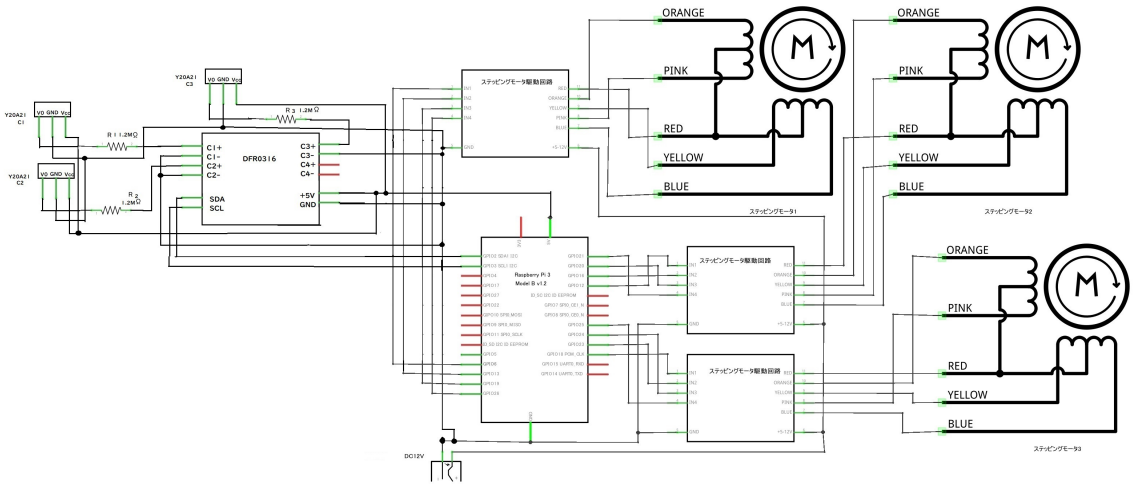

図 2　Rasberry pi と複数ステッピングモータ及び A/D コンバータを接続したステッピングモータカーの配線図

次にソースコード 2 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、自動走行を行った。実行コマンドは python3 ad_convert_drive.py で行った。プログラムのフローチャートは図 3 のとおりである。距離の測定は 3 つのセンサーの測定の際に時間の間隔が必要なため、前進、後退の際のモータのループ処理の間に行う。また、ソースコード 2 で使用するソースコード 3: mcp3424.py を作成した。

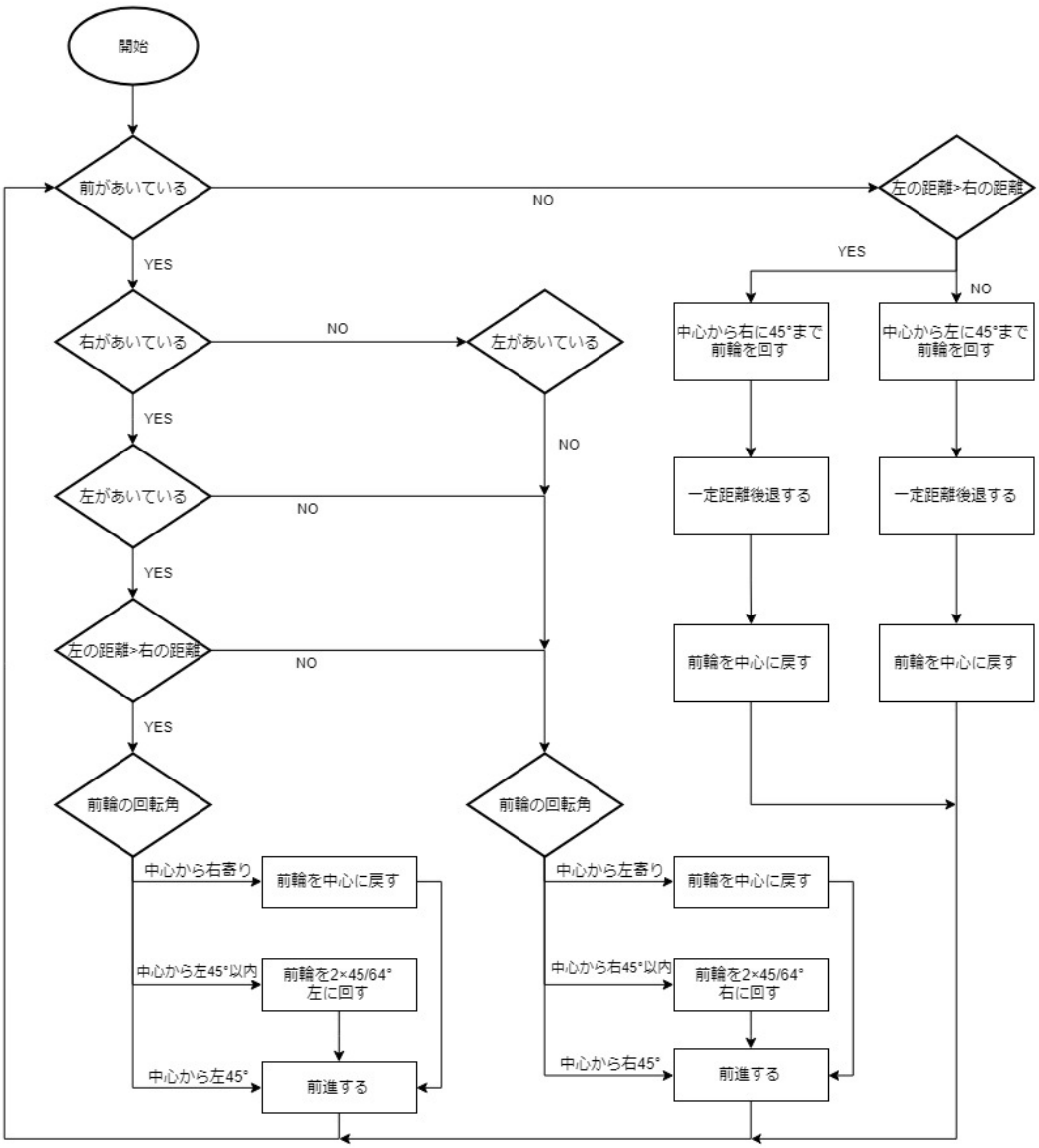

図 3　ad_convert_drive.py のフローチャート

ソースコード 2: ad_convert_drive.py

```
import smbus
import time
import mcp3424
import wiringpi as pi, time

def rotate_rear(reverse, t): #後輪回転用関数
    global ch
```

```
8
9       if reverse == 0:
10          for i in range(4): #正転
11              # センサの電圧を保存
12              data[ch] = i2c.read_i2c_block_data(addr, config[ch], 2)
13              v[ch] = mcp3424.to_volt(data[ch], 12)
14              ch += 1
15              if ch == 3:
16                  ch = 0
17
18              for j in range(4):
19                  if j == i or j == (i+1)%4:
20                      pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
21                      pi.digitalWrite(STEP2_PIN[3-j], pi.HIGH)
22                  else:
23                      pi.digitalWrite(STEP1_PIN[j], pi.LOW)
24                      pi.digitalWrite(STEP2_PIN[3-j], pi.LOW)
25              time.sleep(t)
26
27      else:
28          for k in range(4): #逆転
29              i = (2-k)%4
30              data[ch] = i2c.read_i2c_block_data(addr, config[ch], 2)
31              v[ch] = mcp3424.to_volt(data[ch], 12)
32              ch += 1
33              if ch == 3:
34                  ch = 0
35
36              for j in range(4):
37                  if j == i or j == (i+1)%4:
38                      pi.digitalWrite(STEP1_PIN[j], pi.HIGH)
39                      pi.digitalWrite(STEP2_PIN[3-j], pi.HIGH)
40                  else:
41                      pi.digitalWrite(STEP1_PIN[j], pi.LOW)
42                      pi.digitalWrite(STEP2_PIN[3-j], pi.LOW)
43              time.sleep(t)
44
45  def rotate_front(theta, reverse, t): #前輪回転用関数
46      global ch
47      parse = int(64*theta/45)
```

```
48      if reverse == 0:
49          for turn in range(parse):
50              for i in range(4): #正転（左回転）
51                  data[ch] = i2c.read_i2c_block_data(addr, config[ch], 2)
52                  v[ch] = mcp3424.to_volt(data[ch], 12)
53                  ch += 1
54                  if ch == 3:
55                      ch = 0
56
57                  for j in range(4):
58                      if j == i or j == (i+1)%4:
59                          pi.digitalWrite(STEP3_PIN[j], pi.HIGH)
60                      else:
61                          pi.digitalWrite(STEP3_PIN[j], pi.LOW)
62                  time.sleep(t)
63
64      else:
65          for turn in range(parse):
66              for k in range(4): #逆転（右回転）
67                  i = (2-k)%4
68                  data[ch] = i2c.read_i2c_block_data(addr, config[ch], 2)
69                  v[ch] = mcp3424.to_volt(data[ch], 12)
70                  ch += 1
71                  if ch == 3:
72                      ch = 0
73
74                  for j in range(4):
75                      if j == i or j == (i+1)%4:
76                          pi.digitalWrite(STEP3_PIN[j], pi.HIGH)
77                      else:
78                          pi.digitalWrite(STEP3_PIN[j], pi.LOW)
79                  time.sleep(t)
80
81
82  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
83  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
84  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
85
86  #ステッピングモータ接続端子を出力モードにする
87  pi.wiringPiSetupGpio()
```

```
for i in range(4):
    pi.pinMode(STEP1_PIN[i], pi.OUTPUT)
    pi.pinMode(STEP2_PIN[i], pi.OUTPUT)
    pi.pinMode(STEP3_PIN[i], pi.OUTPUT)

fnear = 1.9 #前方のセンサの物体が近くかどうか判定するための変数（V）
snear = 2.047 #前方のセンサの物体が近くかどうか判定するための変数（V）
T = 0.003 #周期（s）
basic_angle = 0.703125*2 #前輪回転角単位

i2c = smbus.SMBus(1)
addr = 0x6e # デバイスアドレス
config = [mcp3424.cfg_read | mcp3424.cfg_once | mcp3424.cfg_12bit]*3

config[0] |= mcp3424.cfg_ch1 | mcp3424.cfg_PGAx1
config[1] |= mcp3424.cfg_ch2 | mcp3424.cfg_PGAx2
config[2] |= mcp3424.cfg_ch3 | mcp3424.cfg_PGAx2

front = 0
right = 1
left = 2
front_angle = 0

ch = 0 #センサのチャンネル番号
data = [[0,0], [0,0], [0,0]]
v = [0,0,0]
while 1:
    #前輪の回転角の確認
    print(v, front_angle)

    if v[front] < fnear:
        # 前方が一定距離あいている

        if v[right] < snear:
            # 右側が一定距離あいている

            if v[left] < snear:
                # 左側が一定距離あいている

                if v[left] < v[right]:
```

```
                # 左側が右よりあいていている
                # 右側に進む

                if front_angle > -45:
                    # 前輪が中心から左に 45°回されていないときは
                    # 前輪を basic_angle°左に回す
                    rotate_front(basic_angle,0,T)
                    front_angle -= basic_angle

            else:
                # 右側が左よりあいている
                # 左側に進む

                if front_angle < 45:
                    # 前輪が中心から右に 45°回されていないときは
                    # 前輪を basic_angle°右に回す
                    rotate_front(basic_angle,1,T)
                    front_angle += basic_angle

            #一定距離進む
            rotate_rear(0, T)

        else:
            # 左側が一定距離あいていないため
            # 右側に進む

            if front_angle < 0:
                # 前輪が左に回されている場合
                # 前輪を中央に戻す
                rotate_front(abs(front_angle),1,T)
                front_angle = 0

            elif 0 <= front_angle < 45:
                # 前輪が右に回されていて中心からの角度が 45°以内の場合
                # 前輪を Basic_angle°左に回す
                rotate_front(basic_angle,1,T)
                front_angle += basic_angle

            #一定距離進む
            rotate_rear(0,T)
```

```
168         else:
169             # 右側が一定距離開いていないため
170             # 左側に進む
171 
172             if front_angle > 0:
173                 # 前輪が右に回されている場合
174                 # 前輪を中央に戻す
175                 rotate_front(front_angle, 0,T)
176                 front_angle = 0
177 
178             elif -45 < front_angle <= 0:
179                 # 前輪が左に回されていて中心からの角度が 45°以内の場合
180                 # 前輪を Basic_angle°左に回す
181                 rotate_front(basic_angle,0,T)
182                 front_angle -= basic_angle
183 
184             #一定距離進む
185             rotate_rear(0, T)
186 
187     elif v[front] >= fnear:
188         # 前があいていない場合
189         # 左右を見てどちらに下がるか決める
190 
191         if v[left] < v[right]:
192             # 左があいている場合
193             # 中心から右に 45°になるまで前輪を回す
194             rotate_front(45 - front_angle, 1, T)
195 
196             # 一定距離後ろに下がる
197             for _ in range(200):
198                 rotate_rear(1, T)
199 
200             # 前輪を中心の位置に戻す
201             rotate_front(45, 0, T)
202 
203         else:
204             # 右があいている場合
205             # 中心から左に 45°になるまで前輪を回す
206             rotate_front(45 + front_angle, 0, T)
207 
```

```
208             # 一定距離後ろに下がる
209             for _ in range(200):
210                 rotate_rear(1, T)
211 
212             # 前輪を中心の位置に戻す
213             rotate_front(45, 1, T)
214 
215         front_angle = 0
```

ソースコード 3: mcp3424.py

```
1  cfg_read  = 0b10000000
2  cfg_stop  = 0b00000000
3  cfg_ch1   = 0b00000000
4  cfg_ch2   = 0b00100000
5  cfg_ch3   = 0b01000000
6  cfg_ch4   = 0b01100000
7  cfg_once  = 0b00010000
8  cfg_seq   = 0b00000000
9  cfg_18bit = 0b00001100
10 cfg_16bit = 0b00001000
11 cfg_14bit = 0b00000100
12 cfg_12bit = 0b00000000
13 cfg_PGAx1 = 0b00000000
14 cfg_PGAx2 = 0b00000001
15 cfg_PGAx4 = 0b00000010
16 cfg_PGAx8 = 0b00000011
17 
18 volt_ref = 2.048
19 vmax_18bit = 0x01FFFF
20 vmax_16bit = 0x7FFF
21 vmax_14bit = 0x1FFF
22 vmax_12bit = 0x07FF
23 
24 sps_18bit = 3.75
25 sps_16bit = 15
26 sps_14bit = 60
27 sps_12bit = 240
28 
29 def parse_int(data, resolution):
```

```
    # M is MSB since the value style is two's complement
    if resolution == 18:
        # ******MD DDDDDDDD DDDDDDDD
        x = int(((data[0]<<16)&0x030000)|((data[1]<<8)&0x00FF00)|data[1])
        return (-(x&0x020000) | (x&0x01FFFF))

    elif resolution == 16:
        # MDDDDDDD DDDDDDDD
        x = int(((data[0]<<8)&0xFF00)|data[1])
        return (-(x&0x8000) | (x&0x7FFF))

    elif resolution == 14:
        # **MDDDDD DDDDDDDD
        x = int(((data[0]<<8)&0x3F00)|data[1])
        return (-(x&0x2000) | (x&0x1FFF))

    elif resolution == 12:
        # ****MDDD DDDDDDDD
        x = int(((data[0]<<8)&0x0F00)|data[1])
        return (-(x&0x0800) | (x&0x07FF))

def to_volt(data, resolution):
    x = parse_int(data, resolution)
    if resolution == 18:
        return (round(volt_ref*x/vmax_18bit,9)) # LSB: 0.000015625 V

    elif resolution == 16:
        return (round(volt_ref*x/vmax_16bit,7)) # LSB: 0.0000625 V

    elif resolution == 14:
        return (round(volt_ref*x/vmax_14bit,5)) # LSB: 0.00025 V

    elif resolution == 12:
        return (round(volt_ref*x/vmax_12bit,3)) # LSB: 0.001 V
```

2.2.4　作成した多輪駆動車の仕様及び工夫点

作成した車の外観は図 4 の通りである。

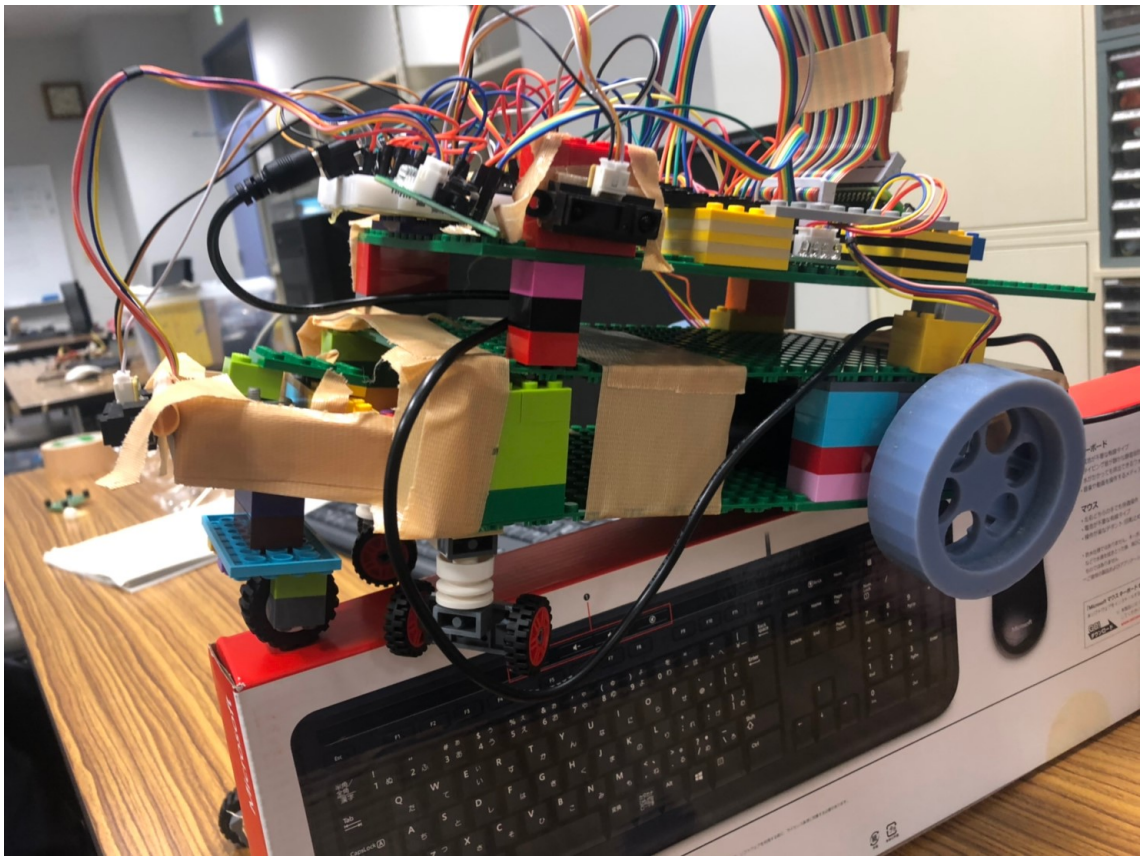

図 4　作成した車

車輪

作成した多輪駆動車は 3 輪駆動式で前輪が方向決定、後輪が前進、後退に用いられる。前方側が方向に比べて上に上がっており、ウィリーのような体勢をとっている。これは、レゴで作成するため、前方と後方を同じ高さにする場合、モータの回転動力を前輪に伝達させるためにはモータを下に向けるためモータの回転を伝える金属部の長さを考えると、車体の上側にモータを設置する必要があり、その場合モータ部から前輪部まで比較的長い距離をブロックで下まで接続してタイヤを付ける必要がある。この時縦に細長いブロックを繋げてモータとタイヤを接続させた場合柱としての強度が足りず、横に長いブロックを繋げてモータとタイヤを接続させた場合柱と車体が前輪の回転の際ぶつかるため車体の向きを変えることができなくなる。そこで、車体の一番下の層にモータを配置し、前輪を繋げるため、前方がある程度後方よりも高くした。

前輪の回転角はフィードバックを行うことができないため、プログラムで前輪の回転角を変数で保持し、前輪の回転の度ステッピングモータが正確な角度で回転しそれが車輪に伝わり、車輪も同角度で回転しているとして、値を変更する。そのため、理想の回転角度と実際の回転角度に誤差が生じた場合に修正を行うことができない。そこで実際の回転角度を正確にするために 2 つのことをした。

回転角度がズレる原因に、前輪が一つであると上からの力が強く摩擦力が大きくなり、そのため前輪を回転させる際にトルクが足りず前輪が回転しないということがあげられる。そのため、1 つめとして、車体前方の左右に 4 のように小さな車輪を付けることで駆動する前輪に加わる力を軽減させた。

また、前輪はモータと下向きで接続させるためモータの回転を伝える金属部に密着に接続しなければ前輪回転時に接続部がズレ、回転が前輪に伝わらなくなり、最悪の場合脱輪を起こす。そこで本実験では図 5、6 のタイヤを作成した。図 5 ではモータに接続できる車輪をガムテープを用いてレゴに密着させ、前輪を実現した。図 6 ではレゴの接続部の円柱状の穴を金属部に無理やり入れることで前輪がモータ回転以外では動かないようにした。

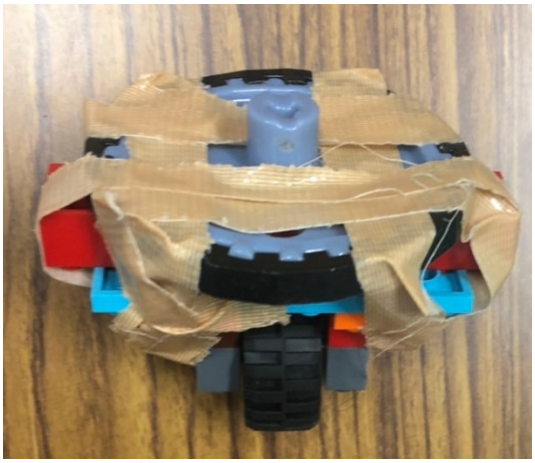

図 5　前輪 1

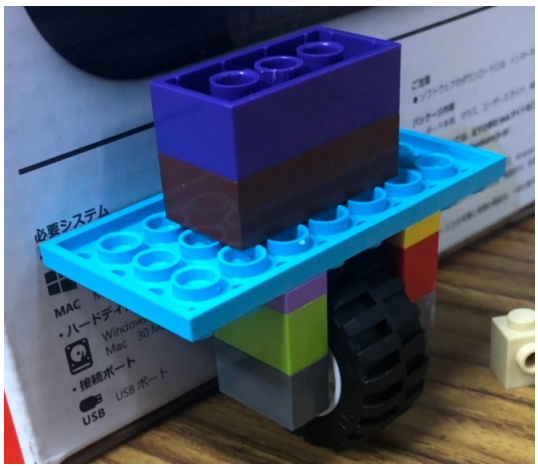

図 6　前輪 2

車輪の駆動及び前進方法

本実験で使用したステッピングモータの回転角は 11.25°であり、ギア比は　64 である。つまり 1 パルスで $\frac{45}{64\times 4}^{\circ}$ の回転をする。この角度は微小であるため前進、後進、前輪の回転を操作する際は　4 パルスを 1 動作として一回の呼び出しでステッピングモータを $\frac{45}{64}^{\circ}$ 回転させた。なお前輪の場合は関数の引数に回転角度を $\frac{45}{64}$ の整数倍でとるようにすることで正確な前輪の回転角を保つようにした。後輪のタイヤの直径は定規による計測で約 6.5cm であることが判明したため、前進、後退の際関数一回の呼び出しで $6.5\times\pi\times\frac{45}{64}\times\frac{1}{360}\simeq 0.039883501cm\simeq 0.04cm$ 動く。そのため前方に壁がない時は $200\times 6.5\times\pi\times\frac{45}{64}\times\frac{1}{360}=\simeq 7.9767cm$ 後ろに下がる。しかし、その際前輪を 45°左右に傾けるため正確な後方の距離ではないが、最低限その空間が存在する場合後方に下がることができる。

車は前方のセンサの電圧が 1.9 以下である場合常に前進するようにプログラムされている。その際左右のセンサの電圧を比べ、壁が遠いほうに前輪を回し前進することでできるだけ壁から遠ざかるようにした。また左右の PGA を× 2 にすることでできるだけ値の変化値を大きくし、空間の判定を早くすることで左右の壁への衝突を防ぐようにした。特にこの車は進行方向側に長い 3 輪駆動であるため内輪差が大きい。そのためできるだけ壁との距離を保つことで右左折の際に壁にぶつかることを防ぐようにした。

赤外線センサ

赤外線センサ 3 つ使い、図 7 の黄色丸の位置に配置した。前方に配置したセンサが前方、左右に配置したセンサがそれぞれ左右の距離を測定する。センサを前方につけることで車の進行方向に対しての左右の障害物の判定を可能にした。

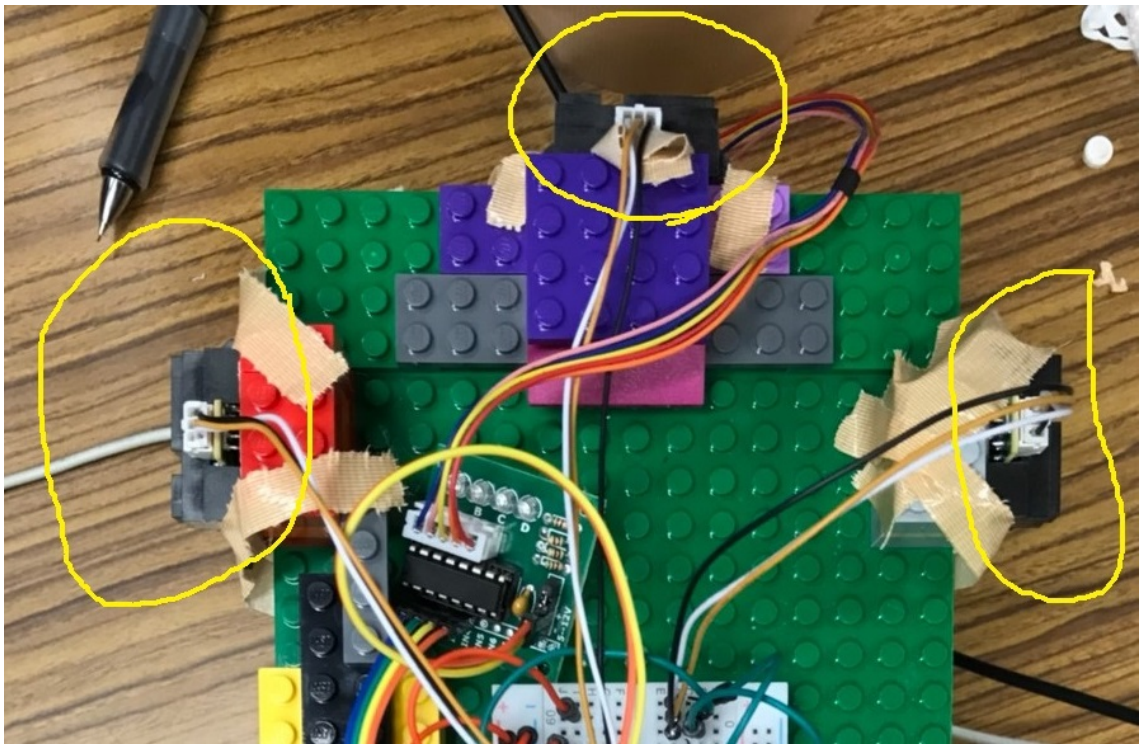

図 7　車の上方 (写真の上方が前方)

2.3　走行結果

指定コースの走破に成功した。走行は蛇行走行であった。プログラムの挙動通りの行動をしたが、左右どちらかの空間があきすぎている場合、想定と異なる動作を行った。また自分たちでいくつかコースを作成したが、狭い袋小路を抜け出すことができなかった。更に、試行を繰り返すうちに前輪が時々変数内の認識している角度と実際の前輪の角度で誤差が生じる場合があった。

2.4　考察

今回の作成した車は前輪が進行方向を決定する役割を担っており、後輪では向きの調節を行わない。そのため、その場での旋回ができず、内輪差及び外輪差が発生し、車の回転には回転方向にある程度の距離を有する。一般に車がハンドルを切って旋回した時に一番外側のタイヤの中心が描く円の半径を最小回転半径という。前輪が 2 つある場合、R(最小回転半径) L (軸距 (前輪の中心軸から後輪の中心軸までの距離))T_f(かじ取り車輪

の輪距 (今回は前輪の幅にあたる)) α (外側車輪のかじ取り角度) β (内側車輪のかじ取り角度) を用いて

$$E = \frac{\frac{L}{sin\ \alpha} + \sqrt{L^2 + (\frac{L}{tan\ \beta} + T_f)^2}}{2}$$

で表される。[2] 本車の場合前輪は一輪であるため α と β を同じ値として考えておおよその値を計算する。精密な計測は行っていないがおおよその値で、L = 20cm、T_f = 2cm、であった。今回旋回する場合前輪は 45 ˚回すので α =45˚で考える。すると、

$$R = \frac{\frac{20}{\frac{\sqrt{2}}{2}} + \sqrt{20^2 + (20+2)^2}}{2} = 29.00820437104946 \simeq 29cm$$

であり、おおよそ 30 センチの最小回転半径を要すことが分かる。つまり、前方に 30cm あれば向きを 90˚回転させることが可能であると考えられる。そのため旋回を考える場合には 30cm 程度の前方の間隔の測定が必要であったと考えられ、実験に使用した far の値では少々不足していると考えられる。

また、今回の実行プログラムでは指定コースは走行できるが、スタートを特殊なスタートポイントから始めることで壁にぶつかる可能性が考えられうる。これは赤外線センサ計の出力電圧の形式を考えると起こりうると考えられる問題である。赤外線センサ計は距離 10cm-80cm では電圧は距離が小さくなるにつれて単調増加する。しかし、80cm 以上遠い場合は電圧は最低値よりも高いある一定の値を出力し、10cm 以上近い場合は電圧は最大値よりも小さい値を出力しその値は距離が小さくなるにつれて単調減少する。このため、配置した車の初期位置が壁から 10cm 以内のかなり狭い幅であるか又は、左のセンサ計の出力した値よりも小さい値を出力する場合このままのプログラムでは壁に衝突しうると考えられる。これを防ぐために、ループ内で進行方向を保持する変数を持ち、その値によって本来増加すべき v の値が減少している場合には壁際 10c 以内にいると判断し前輪を反対方向に回すという動作をすることでこれを回避できると考える。

他に、蛇行しながら前進する際、前輪を左に 45˚になるまで傾けたのち、しばらく走行しても左壁が一定の距離まで近くならない程度の道幅の広さである場合、その場で車が回転し続けると考えられる。そのため車の動作プログラムを改良する必要があると考えられる。また袋小路に車が侵入した際、出られない場合があった。これはプログラム自体で、そのまま後ろにバックするという動作を行う部分がなく、常にどちらかの前に前進することだけを考えてプログラムされているからだと考えられる。そのため前輪を回した後一定距離後ろに下がると壁にぶつかってしまう。もしくは、一定距離下がることができた場合でも、コの字の角が前方になる際、車体が対角線上に存在して、前輪を回して後退すると、左右の距離の近さが反転してしまうため、以降同一の動作を繰り返し続けてしまう。車体の縦幅と横幅に対して、十分に袋小路の空間が存在する場合であるならばこちらも、プログラムを改良し、車の動作アルゴリズムを変更することで脱出することができると考えられる。具体的には迷路の攻略法の一つである右手法を使用することで両者の問題を解決することができると考える。右手法は常に右側の壁に一定の距離を保ちながら進む方法である。具体的には、ソースコード 4: ad_convert_right_hand.py を考えた。このプログラムの rotate_front 及び rotate_rear 関数はソースコード 1 と同じであるため省略する。このプログラムのフローチャートは図 8 のとおりである。このプログラムであれば従来のままでは抜け出すことができない袋小路を抜け出すことができると考える。

また、十分な道幅がある場合であるなら、センサの位置を変え前方と右側面前方及び後方にセンサを付け、右側の 2 つのセンサの値を比較することで車体がどの角度でどの程度壁に近いのかを計算し、右手法を実施することでもまた、走行が実現すると考えられる。

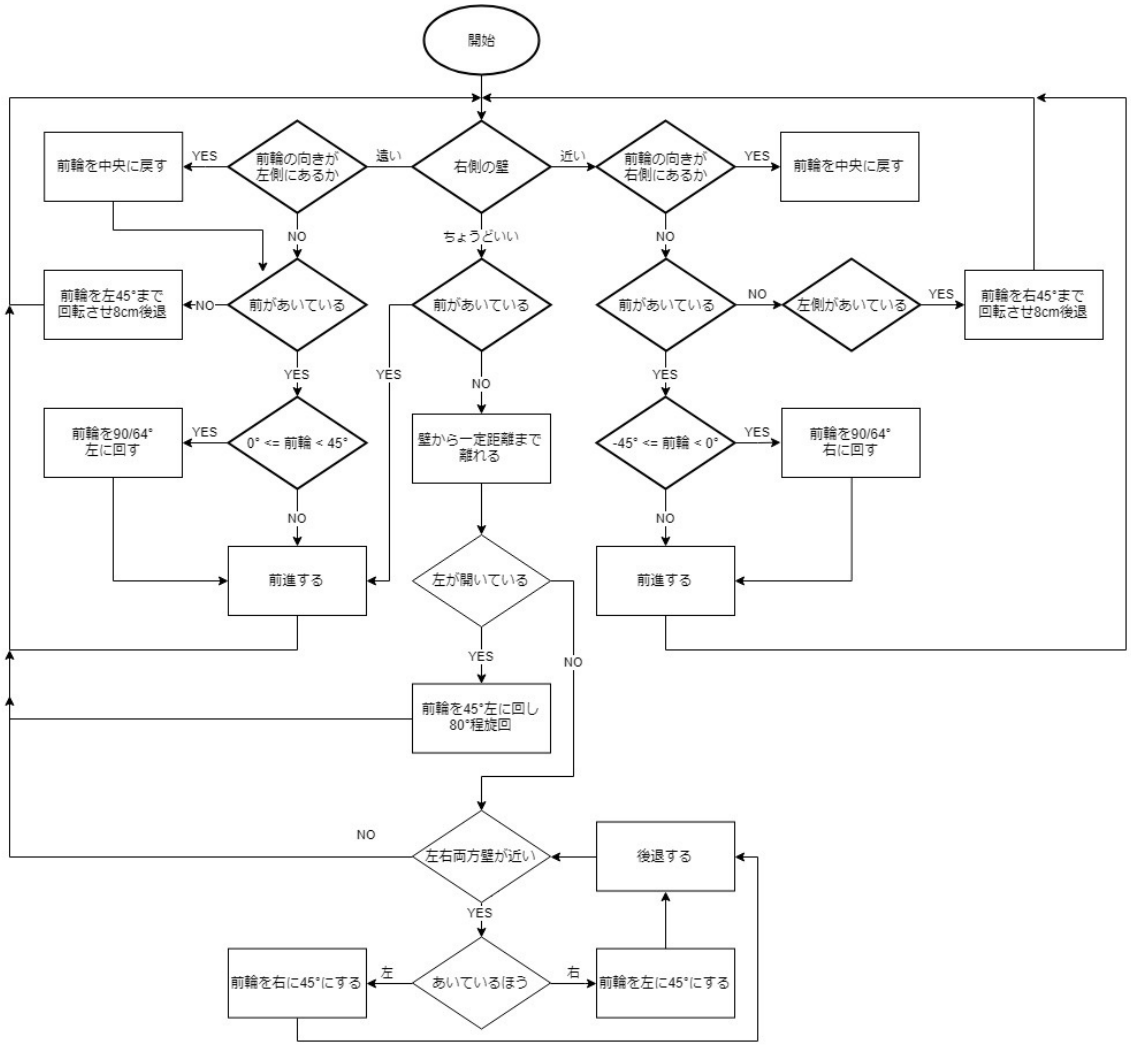

図 8　ソースコード 4 のフローチャート

ソースコード 4: ad_convert_right_hand.py

```
import smbus
import time
import mcp3424
import wiringpi as pi, time

def rotate_rear(reverse, t): #後輪回転用関数
    # （中略）
def rotate_front(theta, reverse, t): #前輪回転用関数
    # （中略）

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
```

```
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
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

#ステッピングモータ接続端子を出力モードにする
pi.wiringPiSetupGpio()
for i in range(4):
    pi.pinMode(STEP1_PIN[i], pi.OUTPUT)
    pi.pinMode(STEP2_PIN[i], pi.OUTPUT)
    pi.pinMode(STEP3_PIN[i], pi.OUTPUT)

near = 1.8 #センサの近くかどうか判定するための変数（V）
far = 1 #センサの遠くかどうか判定するための変数（v）
fnear = 2.0 #前方がバックしなければいけないレベルにいるか判定する変数（V）
ffar = 1.3 #旋回可能なレベルか判定する変数（V）
T = 0.003 #周期（s）

i2c = smbus.SMBus(1)
addr = 0x6e # デバイスアドレス
config = [mcp3424.cfg_read | mcp3424.cfg_once | mcp3424.cfg_12bit |
↪  mcp3424.cfg_PGAx1]*3

config[0] |= mcp3424.cfg_ch1
config[1] |= mcp3424.cfg_ch2
config[2] |= mcp3424.cfg_ch3

front = 0
right = 1
left = 2
front_angle = 0

ch = 0 #センサのチャンネル番号
data = [[0,0], [0,0], [0,0]]
v = [0,0,0]
while 1:

    if v[right] < far:
        # 右側が離れすぎている場合

        if front_angle < 0:
```

```
            # 前輪が左に回されている場合
            # 前輪を中央に戻す
            rotate_front(abs(front_angle),1,T)
            front_angle = 0

        if  v[front] < fnear:
            # 前方が一定距離あいている

            if 0 <= front_angle < 45:
                # 前輪が右に回されていて中心からの角度が 45°以内の場合
                # 前輪を Basic_angle°左に回す
                rotate_front(basic_angle,1,T)
                front_angle += basic_angle

            # 一定距離進む
            rotate_rear(0, T)

        else:
            # バックが必要なレベルで前が近い
            # 中心から左に 45°になるまで前輪を回す
            rotate_front(45 + front_angle, 0, T)

            # 一定距離 (約 8cm) 後ろに下がる
            for _ in range(200):
                rotate_rear(1, T)


    if v[right] > near:
        # 右側が物と近すぎる場合

        if front_angle > 0:
            # 前輪が右に回されている場合
            # 前輪を中央に戻す
            rotate_front(front_angle, 0,T)
            front_angle = 0

        if v[front] < fnear:
            #前方が一定距離以上あいている

            if - 45 < front_angle <= 0:
```

```
            # 前輪が左に回されていて中心からの角度が 45°以内の場合
            # 前輪を Basic_angle°左に回す
            rotate_front(basic_angle,0,T)
            front_angle -= basic_angle

        #一定距離進む
        rotate_rear(0, T)

    else:
    # バックが必要なレベルで前が近い

    if v[left] < near:
        # 左側が十分に空間が開いている

        # 一定距離 (約 8cm) 後ろに下がる
        for _ in range(200):
            rotate_rear(1, T)

    else:
        # 袋小路であると判定
        # 前輪を中心にする
        rotate_front(front_angle, 1, T)
        front_angle = 0

        while v[left] < near and v[right] < near:
            # どちらかの壁がなくなるまで後ろに下がる

            if v[left] < v[right]:
                # 左があいている場合
                # 中心から右に 45°になるまで前輪を回す
                rotate_front(45 - front_angle, 1, T)
                front_angle = 45
            else:
                # 右があいている場合
                # 中心から左に 45°になるまで前輪を回す
                rotate_front(45 + front_angle, 0, T)
                front_angle = -45

            # 後ろに下がる
            rotate_rear(1, T)
```

```
else:
    # 右側に物があるが近すぎない

    if v[front] < ffar:
        #前方があいている
        #一定距離進む
        rotate_rear(0, T)

    elif fnear > v[front] > ffar:
        # 前方に物があり車体を 90°回転させるには
        # 十分な距離がある

        if v[left] < ffar:
            # 左が十分にあいている
            # 前輪を左に中心から 45°になるまで回す
            rotate_front(45 + front_angle, 0, T)

            # 80°ほど旋回するまで前進
            for _ in range(800):
                rotate_rear(0, T)

        else:
            # 袋小路であると判定
            # 前輪を中心にする
            rotate_front(front_angle, 1, T)
            front_angle = 0

            while v[left] < ffar and v[right] < ffar:
                # どちらかの壁がなくなるまで後ろに下がる

                if v[left] < v[right]:
                    # 左があいている場合
                    # 中心から右に 45°になるまで前輪を回す
                    rotate_front(45 - front_angle, 1, T)
                    front_angle = 45
                else:
                    # 右があいている場合
                    # 中心から左に 45°になるまで前輪を回す
```

```
                    rotate_front(45 + front_angle, 0, T)
                    front_angle = -45

                # 後ろに下がる
                rotate_rear(1, T)

    else:
        # 前方に物がある
        # 後ろに (2cm) 下がる
        for _ in range(50):
        rotate_rear(1, T)
```

3　調査課題: I2C 及び A/D-D/A 変換について

3.1　I2C

I2C(アイ・スクエア・シー又はアイ・ツー・シー) はシリアル通信と呼ばれるデータ通信方式の一つであり、シリアル EEPROM メモリなどとの高速通信を実現する方針である。シリアル EEPROM 以外にも表示制御デバイスや、A/D 変換 IC など、I2C インターフェースで接続する物が存在する。 2 本の信号線 SCL(Serial CLock) と SDA(Serial DAta) により、比較的近い場所にあるデバイス間の情報伝達を行うためのシリアルインターフェースである。Standard-mode では最大 100kbit/s、Fast-mode では最大 400kbit/s、Fast-mode Plus(Fm+) では最大 1Mbit/s、High-speed(Hs-mode) では最大 3.4Mbit/s の、8 ビット単位のシリアル双方向データ転送が可能であり、Ultra Fast-mode(UFm) では、単方向、最大 5Mbit/s のデータ転送が可能である。I2C 対応のデバイスは基本的にインターフェースのための回路を内蔵しているため、外付けの部品は必要がなく、また追加削除も 2 本のラインへの接続だけで行うことが可能である。接続関係を図で示すと図 9 であり、複数のデバイスがバス構成で接続される。この形態のため I2C バスと呼ばれる。マスタとはデータ転送を開始し、クロック信号を生成し、データ転送を終了するデバイスのことを言う。スレーブはマスタからアドレス指定されるデバイスである。それぞれ固有のアドレスを持ち、マスタ側からスレーブ側のアドレスを指定して、一対一の送信や受信の指示をする。クロックは必ずマスタから出力され，入力と出力はクロックに同期して行われる。マスタ 1 つに対しスレーブは複数接続が可能である。マルチマスタの場合メッセージを失うことなく、複数のマスターが同時にバスをコントロールするために、同時に通信を始めようとした場合には衝突検出機能とアービトレーション機能によりデータ破壊を防いで共存させることが可能である。それぞれの接続にはプルアップ抵抗が使用される。[5]

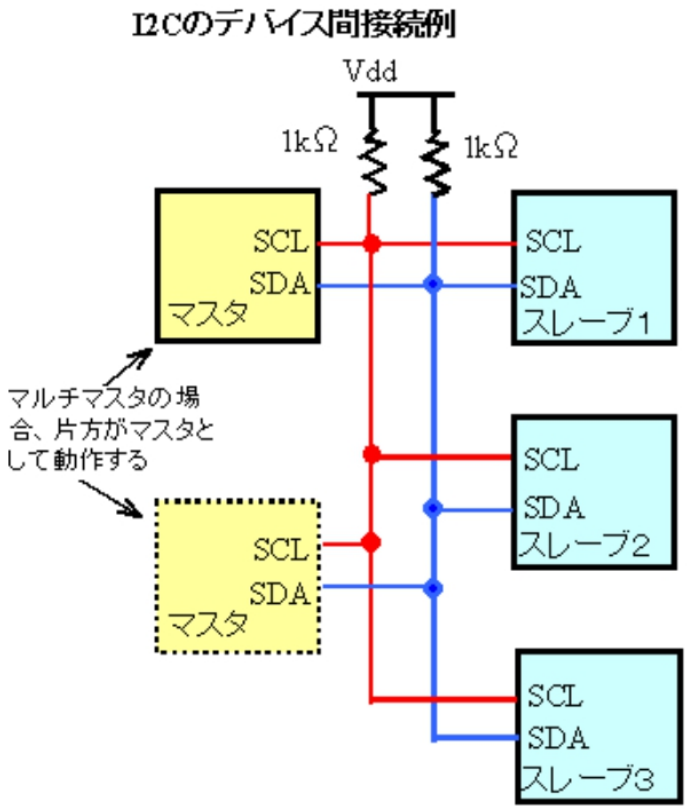

図 9　I2C デバイスの接続例 [4]

I2C の通信は以下の手順により実現される。

1. マスタが通信開始の合図（スタートコンディション）を出す
2. マスタからスレーブアドレスを送信する
3. スレーブからマスタに受信完了の合図（アクノリッジ）を返す
4. データの通信を行う（必要に応じて繰り返す）
5. マスタから通信終了の合図（ストップコンディションを出す

2 のスレーブアドレスを送信する際スレーブアドレスを上位ビットから送信し、マスタから送信する場合は 0 を受信する場合は 1 を最後に付随させる。4 で 8 ビットごとに受信側は受信完了を示す信号を送信する。

以上の通信は SDA で行われる。SDA での通信は SCL のクロック信号と同期しており、例えばスレーブアドレス 0xA0、マスタからスレーブへ 1 バイトのデータ（0x5A）を送信する場合、図 10 のような波形をとる。SCL が H レベルのときに SDA の立ち下がりでスタートコンディション、SCL が H レベルのときに SDA の立ち上がりでストップコンディションとなり、この信号により通信の開始と終了をスレーブ側に知らせる。通信開始の合図（スタートコンディション）を出した後、マスタは通信したいスレーブ側デバイスのスレーブアドレスを送信し、接続されている複数のスレーブが受信する。このうち、アドレスが一致したスレーブのみがスタンバイ状態となり、マスタとの送受信が可能になる。なお、スレーブアドレスが一致しない場合、スレーブ側のデバイスは待機状態へ移行する。SDA と SCL の論理値の判定電圧量は固定ではなく使われる VDD のレベル依存としている。基準 レベルは VDD の 30% および 70%、VIL(Low の規定電圧、これ以下の電圧が

Low と判定される) は 0.3VDD、 VIH(High の規定電圧、これ以上の電圧が High と判定される) は 0.7VDD に設定される。[5]

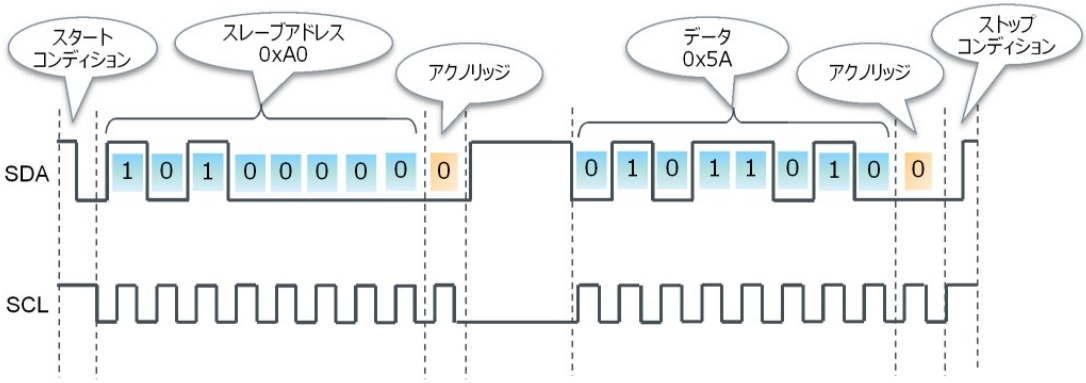

図 10　I2C 通信の例における SDA と SCL の波形 [6]

3.2　A/D-D/A 変換

A/D 変換及び D/A 変換における A、D とはそれぞれアナログ、デジタルを示す。アナログとは値が連続であることを、デジタルとは値が非連続的で離散的であることを指す。温度、圧力、振動 (音波，光波) といったものはアナログな値でありこのアナログな物理量をデジタルな電気信号に変換させるものをトランスデューサ、電気信号を機械的変位量に変換するものをアクチュエータと呼ぶ。アナログ信号をディジタル信号に変換するものを A/D 変換器，ディジタル信号をアナログ信号に変換するものを D/A 変換器と呼び、D/AA/D 変換器は、このアクチュエータやトランスデューサとマイコンとの間を仲介する重要な役目をになう。

3.2.1　A/D 変換

アナログ信号をディジタル信号に変換するには、次の 3 つの手順を行う。

1. 標本化:アナログ信号を一定時間ごとに区切り、その値を読み込む（サンプリングとも呼ぶ）
2. 量子化：標本化し読み込んだ値をディジタル信号に変換できるように小数点以下を切り捨てなどして加工する（量子化による誤差を量子化誤差と呼ぶ）
3. 符号化：量子化された値を指定された 2 進数の桁数で表現する（この 2 進数の桁数を分解能と呼ぶ）

比較器 (コンバータ) は電圧の測定に用いられ宇が、これは、入力電圧を基準電圧と比較して、基準より高いか低いかを出力する回路であり、1bit の AD 変換器とも言える。AD 変換型には例として以下のものがある。

フラッシュ型　nbit 出力の AD 変換では 2^n-1 個の比較器を用いて変換を行う。例えば 0V から 7V まで 1V 間隔で AD 変換して 3bit 出力を得る場合であれば、1V から 7V まで 7 個の比較器を用意して入力と比較を行い、必要応じて出力を 2 進数にエンコードする。

逐次比較型　正確な DA 変換器を用いて入力電圧と比較することで nbit の AD 変換を実現する。比較機を 1 つを繰り返し用いて 1 回ごとに 1bit ごと判定し、n 回の比較を行い正確な nbit 変換を行う。比較の際

に何度も同じ入力を参照する必要があり、その間に入力電圧が変動すると誤変換をする。そのため、変換が完了するまで入力電圧を固定する回路（サンプルアンドホールド回路）が必要となる。

パイプライン型　逐次比較型を直列につなげパイプライン処理をすることで n 段階の回路構成で処理する場合をパイプライン型という。

傾斜型　のこぎり波を常時発生させ、のこぎり波の立ち上がりからのこぎり波の電圧が入力電圧と一致するまでの時間を計測することでデジタル信号に変換する。回路は単純かつ小規模なものになるが処理に時間がかかる。

二重積分型　入力電圧を一定時間充電し、その後基準電圧で放電する際に充電値が 0 になるまでの時間により入力電圧と基準電圧の比を計算し入力電圧を計測する。入力電圧の充電と基準電圧の放電という二つの操作を同じ回路で行い、回路のノイズをキャンセルすることができ、高精度が得やすい。一方、複雑なアナログ回路になり、積分時間がかかることもあり、高速化には向かない。

デルタシグマ方式　デルタ・シグマ方式ではデルタ・シグマ変調という、図 11 のような回路構成であり、入力信号の値と固定電圧値の差を求める Δ（減算）回路、その減算結果を次々に加える Σ（加算）回路、または積分回路、加算結果をある値と比較して大小関係を見つける量子化回路、その量子化回路の出力（デジタル値）に応じて動作するスイッチ回路の 4 つの機能ブロックから構成されたシステムで実現されいる。

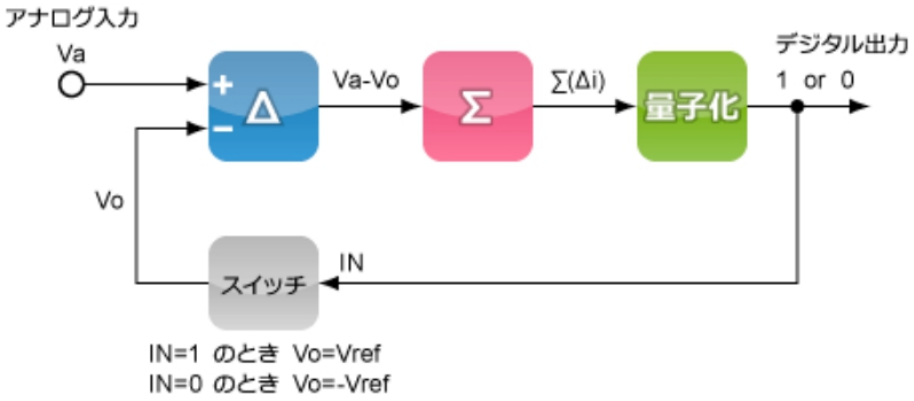

図 11　ΔΣ変調の回路構成及び動作

V_a から入力される電圧と V_0 が減算されその結果が加算回路に足される。加算回路が出力する値により量子化が行われそれに出力されるデジタル値により出力結果及び次クロック時の V_0 の値が変化していく。このように、減算、加算、比較（量子化）を繰り返すことにより、得られる数列の 0 及び 1 の頻度からアナログ信号を再現する。

3.2.2　D/A 変換

デジタル信号をアナログ信号に変換する方法には以下のものがある

1. 抵抗を用いたもの
2. コンデンサを用いたもの

3. 電流源を用いたもの
4. オーバーサンプリング方式

これらを更に使用方法に分けた型について列挙する。

抵抗ストリング型　nbit の場合、2^n 個の抵抗を直列に接続し、基準電圧を分圧し、対応する電圧 2^n 個を全て得る。この中から、デジタル入力に対応する電圧点をアナログスイッチで接続して出力する。

抵抗ラダー型　オペアンプによる演算機能により、抵抗のオンオフや電圧印加から目的の電圧を得る。図 12 の 2 種類の抵抗を組み合わせた回路である R-2R ラダー型抵抗回路によるものが有名である。

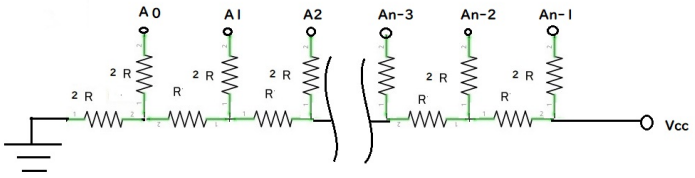

図 12　n-bit R2R 抵抗ラダー型

容量アレイ型　コンデンサをビットに応じた重み付け充電し、全体の電圧を測定する。

電流出力型　抵抗をビットに応じて重みづけこれをスイッチを介して並列接続する。ここに一定の電流をかけて、電流の総和をとることでスイッチが入力を受けたビットに応じて切り替えることで相電流を制御する。結果として 2 進数値に比例する電流が流れる。基本構成はビット数の抵抗とスイッチであり、高速化が容易である。

デルタシグマ型　デジタル入力を時間方向に補完することでサンプリング周波数を数十倍にする。これをデルタ・シグマ変調を行い、低ビットのオーバーサンプリングデータにする。この低ビット出力を DA 変換しローパスフィルタに通すことで折り返し雑音（エイリアス）成分や量子化誤差成分を除去して、アナログ出力とする。

4　まとめ

本実験では赤外線センサ計及び AD コンバータ指定コースを自動で走破可能なステッピングモータカーを作成する実験を行った。赤外線センサ計から出力されるアナログ信号をデジタル信号に変換する AD コンバータの処理、プログラムを用いたデジタル信号値の制御を理解した。また、3 輪駆動車における走行の複雑さを理解した。

参考文献

[1] 赤外線距離センサーを使おう｜おもしろ科学実験室—(閲覧 2019/5/13)https://www.mirai-kougaku.jp/laboratory/pages/161221.php

[2] 車の「最小回転半径」 — Park blog - JAF Mate パーク (閲覧 2019/5/13)https://jafmate.jp/blog/edison/post-19.html

[3] Raspberry Pi で学ぶ電子工作 — 超小型コンピュータで電子回路を制御する著者 — ：金丸隆志—2014 年 11 月 20 日

[4] I2C の基礎知識 (閲覧 2019/5/13)http://www.picfun.com/i2cframe2.html

[5] UM10204 I2C バス仕様およびユーザーマニュアル — NXP Semiconductors(閲覧 2019/5/14)https://www.nxp.com/docs/ja/user-guide/UM10204.pdf

[6] 2-17 I2C とは — ヨースケ先生のラピスセミコンダクタ マイコン豆知識 (閲覧 2019/5/14)http://www.lapis-semi.com/jp/common/miconlp/tips/2-17-i2c%E3%81%A8%E3%81%AF/

[7] ΔΣ型 AD コンバータ — 東芝デバイス＆ストレージ株式会社 (閲覧 2019/5/15) https://toshiba.semicon-storage.com/jp/design-support/e-learning/mcupark/village/ad-converter.html