Raspberry Pi を用いたプログラミング教育

2021 年 2 月 12 日

京都教育大学　理科領域専攻
基礎物理学研究室

171197　久保嘉晴

目次

図目次

1 序論

1.1 はじめに

近年、ICT の普及によって、機械やネットワークの利用がより身近なものになっている。スマートフォンや電化製品をはじめとする電子機器は、もはや日々の生活で、欠かせないものである。その影響か、教育現場でも、新学習指導要領の導入により、小学校では 2020 年度、中学校では 2021 年度から、プログラミング教育が導入される。現在のプログラミング教育では、「プログラミング的思考」の力をつけることを目的としており、授業で本格的なプログラミングを行っているわけではないが、今後さらなる ICT の普及によって、実際にプログラムを作ったりする授業が必修されていく可能性は高い。また、授業で取り扱わなくても、社会に出ていくうえで、プログラミングを用いた研究開発やエンジニアリングといった技術が必要不可欠なものとなってきている。

1.2 本研究について

このような研究開発では、Linux といわれる OS が用いられることが多い。本研究では、Linux 系のコンピュータである Raspberry Pi を用いてプログラミング学習を行う。筆者はこの研究を始めるまでは、プログラミングの経験がなく知識も 0 であった。そのような筆者が Raspberry Pi を用いて一からプログラミング学習してみて感じたことを、今回の研究の結果として捉え、今後発展していくであろうプログラミング教育の教材としての充実性を考えていく。

1.3 Linux について

1.3.1 Linux とは

Linux とは、コンピュータの「脳」のようなものである基本ソフト（OS）の一種である。この Linux は実はたくさんのところで使われている。例えば、現代最も身近にあるスマートフォンに用いられる Android という OS は Linux をベースに作られている。また、Google などの検索エンジン、テレビや DVD レコーダー等多くの機械の OS に Linux が用いられている。つまり、これらの機械について研究し、開発や改良を行っていくためには、この Linux を扱えなければならないのである。

1.3.2 CUI

皆さんがよく使うパソコンや電子機器では、視覚的・直観的に操作を行う、GUI（Graphical User Interface）というインターフェースが使われている。具体的には、タッチパネルやマウスなどを用いて、操作している。これに対し Linux では、言語的・論理的に操作を行う CUI（Character User Interface）が使われている。これは、命令コマンドを打つことによって、ファイル操作やアプリケーションの起動を行うなど、全ての操作をキーボードで行う。一見こちらの操作はややこしく難しいように見える。（というか、実際慣れるまでは操作が難しい。）しかしこの CUI では、いわば「言葉のやり取り」を実現しているため、多様で複雑・繊細な要求をする際に、とても便利となる。

CUI では、図 1 のような「ターミナル」を開き、ここにコマンドを用いて指示を出すことで、コンピュータを動かす。CUI では、「シェル」というソフトウェアがコマンドを解釈することで、やりとりができるように

なっている。

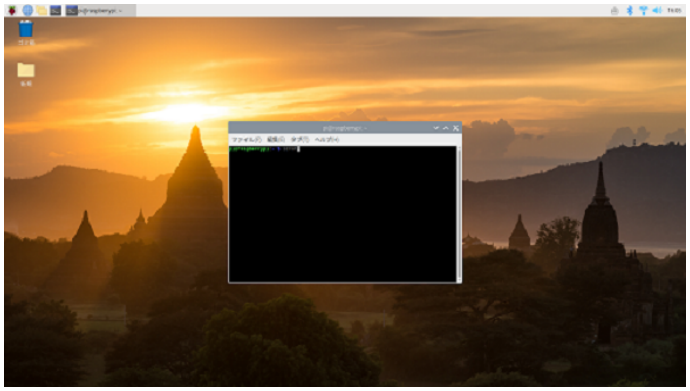

図 1　ターミナルを開いた画面

1.3.3　オープンソース

Linux にはもう一つ、「オープンソース」という大きな特徴がある。開発者がプログラミングを行って完成した成果物を「プログラム」や「ソースコード」という。「オープンソース」とは、この「ソースコード」を秘密にせず完全にオープンにし、さらに誰でも無料で利用や改良できるようにされたシステムである。これにより、そのソフトが広く普及し、それ土台をとして業界が発展した。さらに、不具合を見つけた人が自分で修正したりなど、ソフトウェアをより良いものにすることができるのである。今回の研究でも、このオープンソースという特徴を生かして、学習を行った。

1.3.4　Unix について

ここで、Linux とよく似た Unix について説明しておく。Unix は現存する中で、もっとも古い OS である。1960 年代、アメリカの AT&T 社のベル研究所によって、Multics という OS が開発された。この Multics は画期的なもので、技術革新をもたらしたが、その一方で、たくさんの機能を載せすぎたことにより、想定外に巨大で複雑なものとなり、失敗に終わった。ベル研究所のケン・トンプソンは、この失敗を生かし、機能の少なくて使いやすい OS、UNICS を開発した。multi（複）に対して、uni（単）という意味が込められている。この UNICS は、その後 Unix と改名され、動作の軽さやセキュリティ能力が高いことから、急速に普及していった。現在でも、開発の分野で最前線で活躍している。

この Unix は、当初コードが無料で公開されていた。それにより多くのエンジニアが食いついたのだが、それを使いこなして商売を始める人もたくさん出てきたため、企業としては利益がなくなり収拾がつかなくなった。その結果として、結局 Unix は、ライセンス契約という形となり、オープンソースではなくなった。

そんな中、リーナス・トーバルズというフィンランドの天才大学生が、Unix を参考にして、Linux という OS を作った。リーナス (Linus) が作った、Unix 系の OS なので、Linux と名付けられている。Unix を改良していくのは難しかったので、なんとゼロから作り上げたのである。Unix については参考にしているだけなので、企業のソースコードを用いているわけではなく、著作権に引っかかることもなかった。よって見た目や機能は Unix に似ているのだが、中身は完全にリーナスのオリジナルである。さらにリーナスは、この Linux のコードをオープンソースにし、だれでも改良できるようにした。これにより、エンジニアに大きな衝撃が走

り、一気に普及されていった。

このように Unix と Linux の一番の違いは、オープンソースであるかどうかという点であるが、Unix もいまだに使いやすさなどから普及され続けている。

1.3.5 実際に Linux を使うには

では、実際に Linux を用いて学習していきたいのだが、現代我々が利用しているパソコンにはすでに Windows や Mac OS のような OS が入っているのもが多い。そのため、Linux の OS を入れて使うためには、新しく空のハードディスクを買って入れ替えるか、デュアルブート（１つのパソコンに複数の OS を同居させ、起動させるたびにどちらを利用するか選択する仕組み）を用いる必要がある。しかしこれらの方法はかなり難解であり、慣れていない人が行うと、パソコンそのものが壊れたり、トラブルを起こす可能性がある。よって、実際に Linux を使うには、Linux 専用のコンピュータを用意するのが一番得策なのである。

1.4 Raspberry Pi について

1.4.1 Raspberry Pi とは

本研究で用いる Raspberry Pi とは、2012 年にイギリスのラズベリー財団で開発された名刺サイズのシングルボードコンピュータである。このボードには、コンピュータとしての最低限の機能が搭載され、キーボードやマウス、ディスプレイと接続することで、通常のコンピューターのように利用することができる。

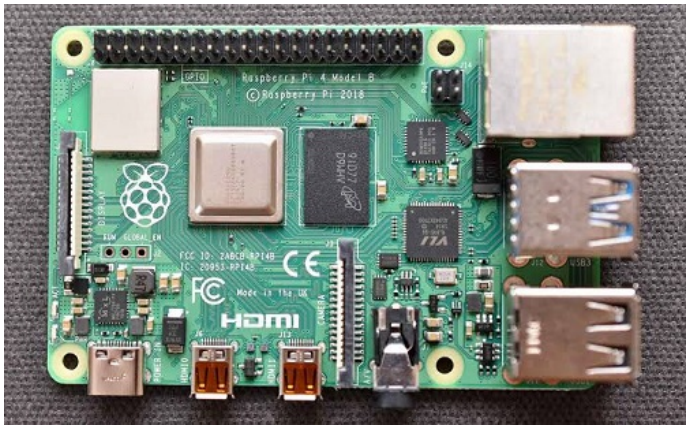

図 2　Raspberry Pi の全体図

この Raspberry Pi 上で動作する OS にはいくつか種類があるが、最も利用されているものは Raspbian という OS であり、これは Linux 系の OS である。「1.3.3. 実際に Linux を使うには」で述べたように、Linux を使うには、Linux 専用のコンピュータを用意する必要がある。しかし、そのために Linux の入っているパソコンを購入するのは、かなりのコストがかかる。これに対し、この Raspberry Pi は本体価格が 5000 円前後であり、かなり安価に手に入れることができる。そのため、Linux 系の OS を使った学習を行うのに、この Raspberry Pi が最適であると感じ、今回の研究で用いることになった。

1.4.2 Raspberry Pi の利点

Raspberry Pi の利点としては、安価に Linux 系の OS を用いて学習できることのほかに、プログラミング言語である Python との親和性が高いことがある。機械学習のプログラムを書く際の言語として Python が使われることが多い。そこでこの Python との相性が良いことがもう一つの利点として挙げられる。実際、Raspberry Pi の Pi は Python の読みの一部「パイ」からきている。

1.5 Raspberry Pi の準備

1.5.1 Raspberry Pi の接続

本研究では、SUNFOUNDER の PiCar-4WD というロボットカーのキットに Raspberry Pi を取り付けることで、主にこのロボットカーを動かしたり、そのプログラムについて考える学習をした。

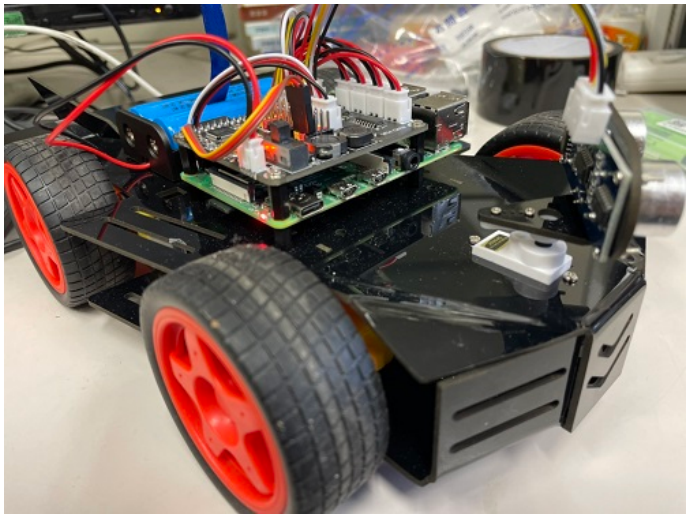

図 3　Raspberry Pi を取り付けた様子

さらにそこに、ディスプレイ、キーボード、マウス、電源を図 4 のように取り付けることで作業を行った。

図 4　Raspberry Pi とデバイスの接続

1.5.2　Raspbian のインストール

本研究では、Raspberry Pi に Raspbian という OS を次の過程でインストールした。このインストール過程は、PiCar-4WD に付属していた電子説明書に従った。

1. Windows のパソコンに NOOBS というインストーラーを Raspberry Pi の公式サイトからダウンロードした。このとき、NOOBS の他に、NOOBS Lite というインストーラーを用いることができる。NOOBS は一式ファイルの中に、Raspbian も同梱されている。そのため、Raspbian をオフラインの状態でもインストールすることができる。一方、NOOBS Lite には Raspbian が同梱されていないため、ダウンロードにかかる時間は短縮される。しかし、Raspbian をインストールする際には、ネット環境が必要となる。今回はどちらのインストーラーを用いても可能であったため、NOOBS の方を用いた。
2. Windows のパソコンに SD カードを差し込み、「SD Formatter」というソフトウェアを用いて、SD カードをフォーマット（初期化）した。これにより、SD カードの性能をフルに引き出せる。
3. ダウンロードした NOOBS フォルダー内のファイルをすべて SD カードへコピーした。
4. SD カードを Raspberry Pi に挿入し、Raspberry Pi を起動させると、Raspbian のインストールの画面が出てきたため、インストールした。
5. インストールが完了すると、自動的に再起動が起こり、Raspberry Pi のデスクトップ画面が出現した。その後、初期設定の画面が出てきたため、国、言語、タイムゾーンを設定した。パスワードを設定する画面も出てきたが、この時点では設定しなかった。設定しなかった場合、パスワードはデフォルトの「raspberry」が適応される。最後に Wi-Fi の設定画面が現れたため、研究室使用している Wi-Fi に接続した。

これで、Raspbian のインストールや初期設定が終わり、コンピュータとして使うことができるようになった。

2　Raspberry Pi を用いたプログラミング学習

2.1　プログラムの取り込み

Raspberry Pi の準備が整ったところで、ロボットカーを動かしてみる。このロボットカーには、サンプルプログラムがもともと用意されており、そのコードファイルが GitHub にのせてあるため、次のコードを用いた。

```
$ cd /home/pi/
$ git clone https://github.com/sunfounder/picar-4wd
```

これにより、サンプルのプログラムが GitHub から複製された。

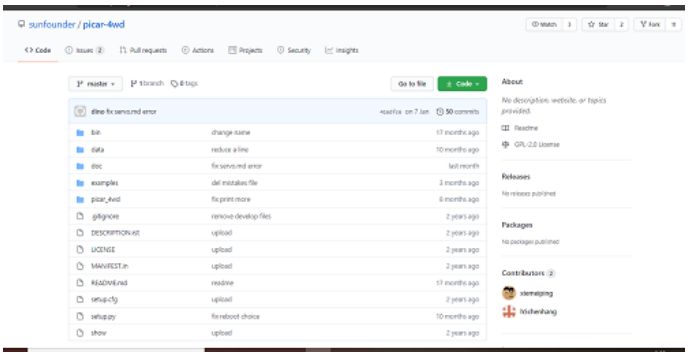

図 5　GitHub の公式ページ

2.2　プログラミングの実行

複製したプログラミングを実際に実行した。これは、picar-4wd の examples のフォルダに入っているので、次のコードでディレクトリを変更した。

```
$ cd /home/pi/picar-4wd/examples
```

ここには python3 というコマンドがあり、そのコマンドから Python のプログラムを使うことができる。サンプルの Python によるサンプルのプログラムは 4 種類あった。

2.2.1　キーボードコントロール

```
$ python3 keyboard_control.py
```

というコードで実行できる。これは、Raspberry Pi に接続したキーボードによって操作するというものである。このプログラムにプログラミングされていたのは、

1.「W」を入力→前進
2.「S」を入力→後退

3.「D」を入力→右折
4.「A」を入力→左折
5.「6」を入力→スピードアップ
6.「4」を入力→スピードダウン
7.「Q」を入力→プログラムの終了
8. これら以外のキーを入力→停止

という操作方法であった。右折と左折は、その場で向きを変える動きであった。スピードは 0〜100 までの段階を 10 段階ずつ調整することができた。

2.2.2 障害物回避

```
$ python3 obstacle_avoidance.py
```

というコードで実行できる。これはロボットカーの先にある Ultrasonic(超音波) というセンサーによって、障害物を感知すると、回避する動きができる。プログラムを実行した瞬間から、何もないときは常に前進する。センサーによって障害物を感知したら一度停止し、その後右折し、また前進する。これを、プログラムを終了するまで続ける。プログラム実行中は常に、Ultrasonic センサーが、右 90° から左 90° の 180° を往復している。このプログラムは、Ctrl+C で終了することができる。

図 6　Ultrasonic センサー

2.2.3　床の黒いラインをたどる動き

```
$ python3 track_line.py
```

というコードで実行できる。ロボットカーの下についている grayscale というセンサーによって、黒いラインをたどって動くことができる。このセンサーは、白から黒までの明るさを感知することで、色を判断することができる。（白から黒で判断するため、赤などは不可能である。）実際床に黒色のカラーテープを貼り、プログラムを実行したところ、そのテープをたどって動いた。途中にテープの道をカーブさせると、ロボットカーもそれに対応してカーブをした。テープの端まで行き、道がなくなると、その場で回転した。180° 回転したところで、来た道を折り返した。

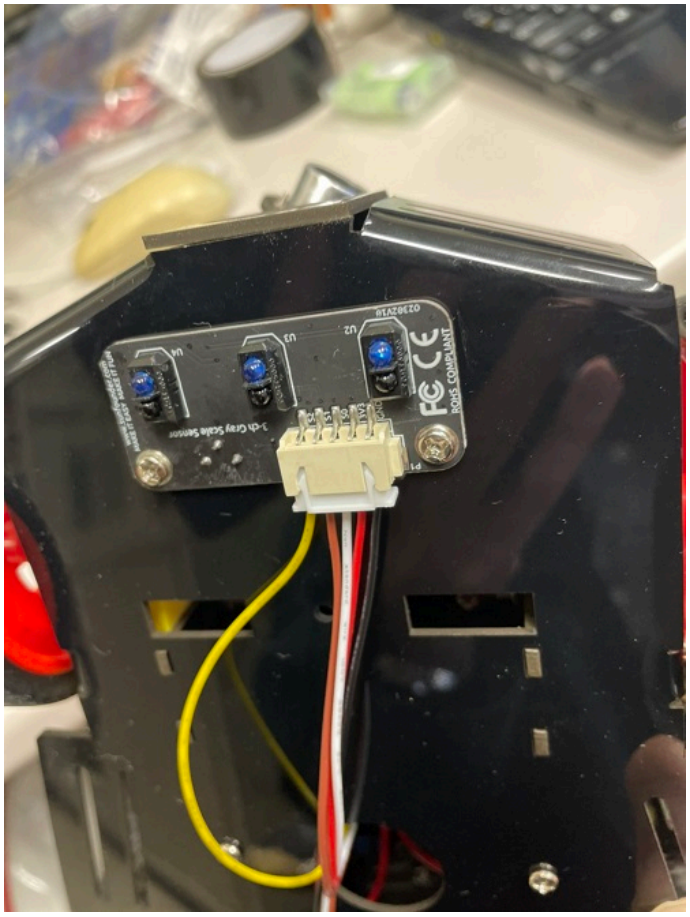

図 7　grayscale センサー

2.2.4　追跡する動き

```
$ python3 follow.py
```

というコードで実行できる。これは障害物回避のときと同じ Ultrasonic センサーによって物体を感知し、その物体を追跡する動きができる。実際にロボットカーの前を歩くと、ちょっと追いかけていくような動きがあったが、すぐに止まった。センサーの位置もあり、反応しにくいみたいであった。センサーの目の前に手を差し出して動かすと、一番動かしやすかった。しかしそれでも、ある程度手を追いかけると、途中で停止した。手を近付けると、今度は少し後退し、その後再び手を追いかけた。このプログラムでロボットカーを自在に操るのは難しく、センサーが過剰に反応してしまっている動きをしているような印象を受けた。

2.2.5　ウェブエグザンプル

これらのプログラムとは別で、ウェブエグザンプルというプログラムもある。これは Python によるプログラムではないため、他の 4 つとは変わってくる。

```
$ cd /home/pi/picar-4wd/
$ picar-4wd web-example
```

まずこのコードを実行させる。すると、図 8 のように、画面に色々な情報が出てくる。

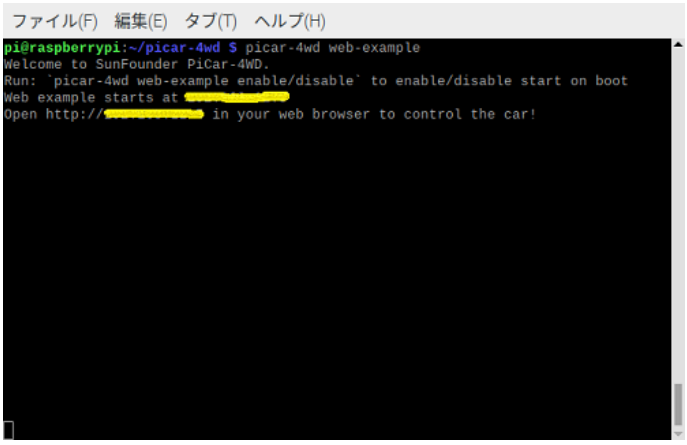

図 8　ウェブエグザンプルの実行画面

この中に、IP アドレスがのっている部分（図の黄色い線の部分）があるため、それをスマートフォンやパソコンのブラウザに打ち込んで検索すると、そのブラウザに図 9 の画面が表示される。

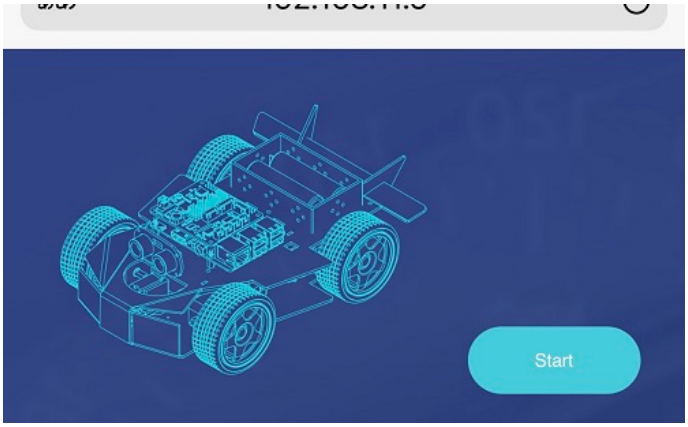

図 9　ウェブコントロールのスタート画面

この画面のスタートを押すと、図 10 のように、リモコンのような画面が表示される。これにより、ロボットカーをコントロールできる。

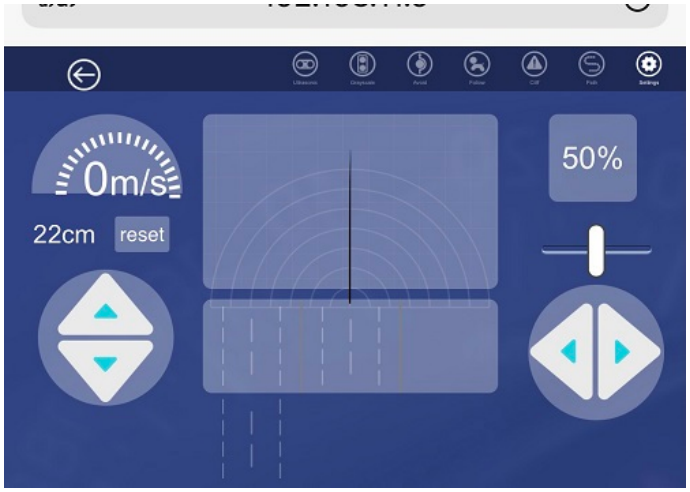

図 10　ウェブコントロールの操作画面

このプログラムでは、これまでのプログラムと違って、リモコン操作によってスムーズにロボットカーを動かせる。また、ただロボットカーを動かすだけでなく、Ultrasonic センサーや grayscale センサーの機能、タイヤの回転を確認することができる。Ultrasonic センサーの確認は、図 11 のように、センサによって感知した物体の位置を、赤い点で示している。(モニターと赤い点がずれているのは、単にバグかと思われる。)

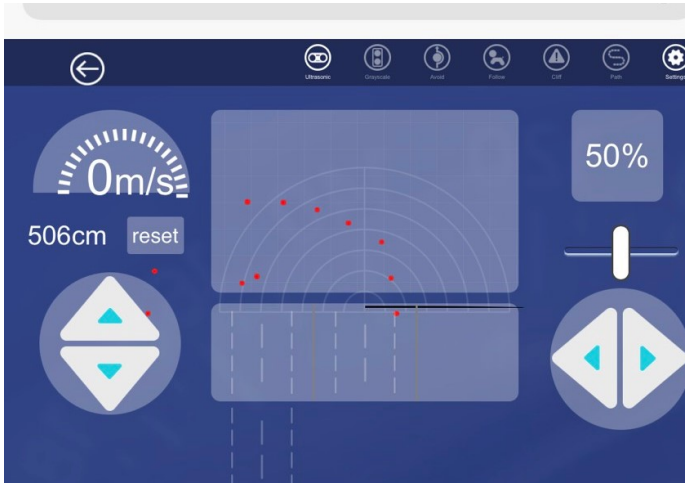

図 11　Ultrasonic センサーの動作確認画面

このウェブコントロールは、Python によってつくられたプログラムではない。コードの実行後、ブラウザに IP アドレスを打ち込むことで、このブラウザが、web インターフェースにアクセスできるようになる。Raspberry Pi がサーバ、ブラウザがクライアントとなるわけである。これにより、ロボットカーを直接コントロールできるのである。

2.3　SSH 接続によるリモートコントロール

Raspberry Pi では SSH という機能を用いることができる。これは、1 つのネットワークを利用して、ネットワーク上に存在するサーバにアクセスし、パソコンからサーバ操作を行う機能である。いわゆる遠隔操作である。仕組みとしては、ネットワークを仲介人として、パソコンと Raspberry Pi でやり取りを行うと考えてよい。今回は、研究室にある同じ Wi-Fi に Raspberry Pi とパソコンの両方を接続することで利用した。SSH では、全く知らない人からアクセスされ操作される危険性もあるため、初期設定では、SSH の機能が無効化されていることが多い。今回も Raspberry Pi の設定画面から SSH を有効化にしてから作業をした。Windows には、Linux のようなターミナルがない。そこで、SSH 接続してターミナルで指示を出すためのソフトウェアである「PuTTY」をインストールして利用した。これを開くと図 12 のような画面が出てくる。

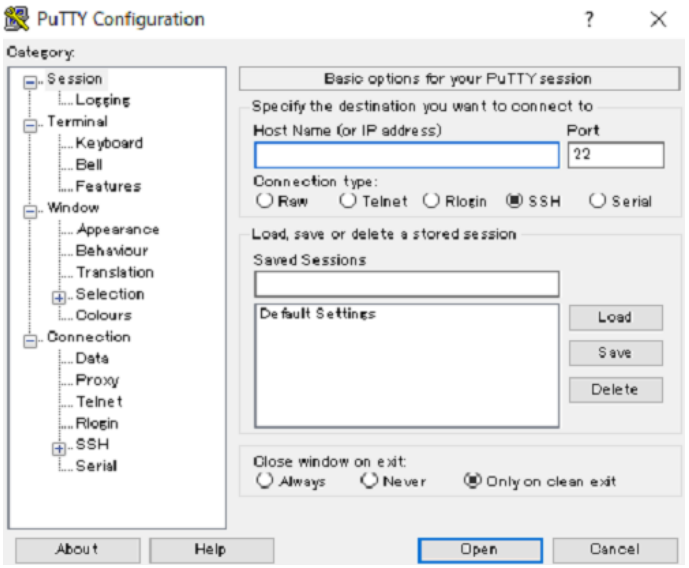

図 12　PuTTY の画面

この画面の IP アドレスを入れる欄（図で青枠になっている部分）に Raspberry Pi が使っているネットワークの IP アドレスを入力した。IP アドレスは以下のコードによって調べることができる。

```
$ ifconfig
```

これにより、パソコンから Raspberry Pi にリモートで指示を出すことができる。有線のキーボードを接続した状態では、作業やコードを実行する際に邪魔になるので、こちらのリモート操作の方が圧倒的に楽である。実際に本研究でも、常に用いていた。

2.4　プログラムの解読

各コードのプログラムは vi エディタによってみることができる。（そもそもはプログラムを編集するためのもの。）これを開くためには、

```
$ vi プログラム名
```

で開ける。例えばキーボードコントロールの場合は

```
$ vi keyboard_control.py
```

となる。開くと図 13 のような画面になる。

図 13　vi エディタを開いた時の図

これにより、Python によるもののそれぞれのプログラムを開き、少し解読してみた。Python についての知識はまだまだ不十分であるため、コードの全てを解読するのは難しく、憶測で考えている部分もある。

2.4.1 キーボードコントロール

キーボードコントロールのプログラムは下のようなものであった。

```
1  import picar_4wd as fc
2  import sys
3  import tty
4  import termios
5  import asyncio
6
7  power_val = 50
8  key = 'status'
9  print("If␣you␣want␣to␣quit.Please␣press␣q")
10 def readchar():
11     fd = sys.stdin.fileno()
12     old_settings = termios.tcgetattr(fd)
13     try:
14         tty.setraw(sys.stdin.fileno())
15         ch = sys.stdin.read(1)
16     finally:
17         termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
18     return ch
19
20 def readkey(getchar_fn=None):
21     getchar = getchar_fn or readchar
22     c1 = getchar()
23     if ord(c1) != 0x1b:
24         return c1
25     c2 = getchar()
26     if ord(c2) != 0x5b:
27         return c1
28     c3 = getchar()
29     return chr(0x10 + ord(c3) - 65)
30
31 def Keyborad_control():
32     while True:
33         global power_val
34         key=readkey()
35         if key=='6':
36             if power_val <=90:
37                 power_val += 10
38                 print("power_val:",power_val)
39         elif key=='4':
40             if power_val >=10:
41                 power_val -= 10
42                 print("power_val:",power_val)
43         if key=='w':
44             fc.forward(power_val)
45         elif key=='a':
46             fc.turn_left(power_val)
47         elif key=='s':
48             fc.backward(power_val)
49         elif key=='d':
50             fc.turn_right(power_val)
51         else:
52             fc.stop()
53         if key=='q':
54             print("quit")
55             break
56 if __name__ == '__main__':
57     Keyborad_control()
```

まずこのコードの 1〜5 行目で、「モジュール」のインポートを行っている。Python で用いる関数には、インストールした際に、初めから組み込まれているものとそうでないものがある。このような、組み込まれていない関数は、「ライブラリ」の中にあるため、これを使うためには読み込む必要がある。ライブラリは、モジュールという単位で分類してまとめてあるため、これを読み込む。Python の利点の一つとして、このライブラリの豊富さがある。このプログラムでインポートしたモジュールについて、簡単なイメージで説明する。

1 行目では、picar-4wd というモジュールに fc という名前を付けてインポートしている。このモジュールは、後に「まっすぐ進む」や「右に曲がる」などの行動を指示するときに使われているため、指示通りに動く

ようにタイヤのモーターの回転速度や回る向きを調製するモジュールだと考えられる。その後の sys というモジュールはシステムパラメータに関係しており、tty モジュールと termios モジュールは端末の制御に関するものである。asyncio モジュールでは、非同期処理（並行処理）を可能にすることができる。

7 行目以降で、プログラムを行うための準備が行われている。ざっくり説明すると、スピードの設定や、押したキーボードをコマンドとして認識するための設定を行っている。キーの設定には、readchar や readkey を使用してキーの入力状態を取得し、if 文を用いていくつか関数を割り当てている。

31 行目からは、入力したキーについて設定されている。ここで、先ほど fc と名前をつけたモジュールが使われている。forward や turn_right のように前後左右の動きをするための関数が定義されており、() 内にスピードを指示している。今回では、power_val という変数を用いているため、スピードを変化させた際に、これらの動きにも適応されている。スピードを上げたり下げたりするコマンドには、power_val が 90 以下や 10 以上のときという条件定義がされているため、調製できる範囲が 0〜100 までとなっている。キーボードコントロールのコードはこのような仕組みになっていた。

2.4.2 障害物回避

```
1  import picar_4wd as fc
2
3  speed = 30
4
5  def main():
6      while True:
7          scan_list = fc.scan_step(35)
8          if not scan_list:
9              continue
10
11         tmp = scan_list[3:7]
12         print(tmp)
13         if tmp != [2,2,2,2]:
14             fc.turn_right(speed)
15         else:
16             fc.forward(speed)
17
18 if __name__ == "__main__":
19     try:
20         main()
21     finally:
22         fc.stop()
```

このコードでも、インポートから入っている。ここでは、picsr-4wd のモジュールのみインポートしている。おそらく今回は、一度コードを実行した後は、他のコマンドを打ち込むようなことがないからであると考えられる。

5 行目から、Ultrasonic のセンサーで感知したことについての設定を行っている。Ultrasonic センサーは、サーボモータによって前方 180° の範囲を往復しながら感知できる。測定する際は、180° を 18° ずつ、計 11 回に分けて測定している。このうちの 3 番から 6 番の 4 つの角度について、それぞれ物体があるか判断した結果を、tmp というリストに並べるように指示している（11 行目）。各角度それぞれで、測定結果が「0,1,2」の三段階の値で送られる。センサーが障害物を感知しなかったときは、「2」という値を返す。ここで、tmp リストの値が、4 つとも「2」と送られた場合以外は右へ曲がり、そうでないとき（全て 2 が送られた時）はまっすぐ進むというコードとなっている。コード実行中は、図 14 のようにリストが随時表示される。このリストが [2,2,2,2] となるときはまっすぐ進み、それ以外 ([0,0,1,2] や [1,1,1,2] など) のときは右折する。

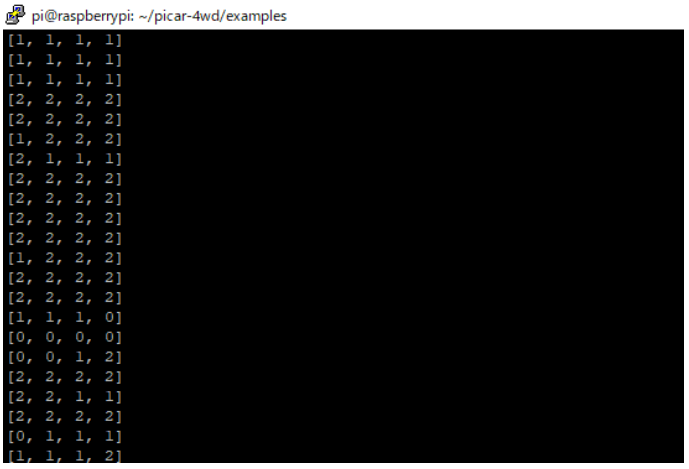

図 14　障害物回避実行中の画面

このプログラムでは、進行指示の後ろに、power_val ではなく、speed となっている。これはプログラムの途中で変更されることがないので、定数となる。

2.4.3 床の黒いラインをたどる動き

```
1  import picar_4wd as fc
2
3  Track_line_speed = 20
4
5  def Track_line():
6      gs_list = fc.get_grayscale_list()
7      if fc.get_line_status(400,gs_list) == 0:
8          fc.forward(Track_line_speed)
9      elif fc.get_line_status(400,gs_list) == -1:
10         fc.turn_left(Track_line_speed)
11     elif fc.get_line_status(400,gs_list) == 1:
12         fc.turn_right(Track_line_speed)
13
14 if __name__=='__main__':
15     while True:
16         Track_line()
```

これも同様に、最初にモジュールのインポートを行い、その後スピードを設定している。今回はスピードを Track_line_speed という定数で定義している。5 行目以降では、garyscale センサーで判断した色をリストにし、そのリストの数字によって指示をしている。このリストの数字がよくわからないが、数字が 0 のときにまっすぐ進む指示がされているので、黒色を感知したときだと思われる。そのほかで、数字が-1 となったときは、左に曲がる、1 となったときは右に曲がるようになっており、黒色を感知できなくなった（黒いラインが途切れた）ときであると考えられる。

2.4.4 追跡する動き

```
import picar_4wd as fc

speed = 30

def main():
    while True:
        scan_list = fc.scan_step(23)
        # print(scan_list)
        if not scan_list:
            continue

        scan_list = [str(i) for i in scan_list]
        scan_list = "".join(scan_list)
        paths = scan_list.split("2")
        length_list = []
        for path in paths:
            length_list.append(len(path))
        # print(length_list)
        if max(length_list) == 0:
            fc.stop()
        else:
            i = length_list.index(max(length_list))
            pos = scan_list.index(paths[i])
            pos += (len(paths[i]) - 1) / 2
            # pos = int(pos)
            delta = len(scan_list) / 3
            # delta *= us_step/abs(us_step)
            if pos < delta:
                fc.turn_left(speed)
            elif pos > 2 * delta:
                fc.turn_right(speed)
            else:
                if scan_list[int(len(scan_list)/2-1)] == "0":
                    fc.backward(speed)
                else:
                    fc.forward(speed)

if __name__ == "__main__":
    try:
        main()
    finally:
        fc.stop()
```

同様に最初にインポートをし、定数を定義してスピードを設定している。このプログラムには解読するのがかなり難しかった。ユーザーマニュアルによると、このプログラムでは Ultrasonic 測定した 11 の値をすべてリストに入れている。例えば右前に何か物体があるとき [2,2,2,2,2,2,1,1,1,2,2] といったようなリストになる。このリストの内の「2」を区切りにして新しくリストを作る。この例では、[1,1,1] というリストが作られる。これは前のリストの 7〜9 番目であるため、角度として、108° から 144° となる。この範囲に物体があると感知しており、この角度の中央値を取る。そしてこの物体との距離が 10cm になるように保っている。

しかし、10cm を保つための動きをするコード解読するのがかなり難しい。作られたリストの 1 つに length_list というものがあり、このリストの最大値が 0 となったとき、ストップする。その後、最終的に pos という関数と delta という関数ができている。delta という関数は、「スキャンしたリストの要素数/3」となっているが、pos についてはあまりわかっていない。(「2」を区切りとして作ったリストと関係しているようである。) この 2 つの関数の関係が、

$$\text{pos} < \text{delta}$$

となるとき、左に曲がり、

$$\text{pos} > 2 \times \text{delta}$$

となるとき、右に曲がる。

32 行目からの関係式もわかりにくいが、絶対値が関係している。このときの絶対値は Ultrasonic で感知している範囲の半径ではないかと推測される。この関係式が 0 となるとき, 後退し、それ以外のときは前進する仕組みとなっている。

2.5　プログラムの書き換え

先に述べたようの、vi エディタでは、プログラムの書き換えを行うことができる。おそらくプログラミングと聞いて一般に思いつくのはこの書き換えであると思う。そのため、私自身もプログラムの書き換えを少ししてみた。書き換えるためには、Python のコードがわかっていないといけないため、現在理解できている部分のみを書き換えた。

vi エディタでは、一般的なテキストエディタとは異なり、入力モードとコマンドモードの 2 つがある。これらを随時切り替えながら、編集を行っていかなくてはならない。入力モードでは、一般的なテキストエディタのように文字を入力することができる。しかし、文字を消したり、カーソルを移動したりするときには、随時コマンドモードへの切り替えが必要である。今回使用したコマンドをまとめると、図 15 のようになる。

コマンド	処理
x	カーソル位置の文字を削除
dd	カーソル位置の行を削除
i	カーソル位置で入力モードに切り替え
Esc	コマンドモードに切り替え
:w	保存
:w ファイル名	名前を付けて新しく保存
:q	終了
:wq	保存して終了

図 15　vi エディタで使用したコマンド一覧

このように、vi エディタは操作方法が特殊であり、慣れるまでに時間がかかった。

2.5.1　キーボードコントロールの書き換え

vi エディアを用いて、キーボードコントロールのキーをそれぞれ、次のように書き換えてみた。

```
import picar_4wd as fc
import sys
import tty
import termios
import asyncio

power_val = 50
key = 'status'
print("If you want to quit.Please press q")
def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def readkey(getchar_fn=None):
    getchar = getchar_fn or readchar
    c1 = getchar()
    if ord(c1) != 0x1b:
        return c1
```

```
25      c2 = getchar()
26      if ord(c2) != 0x5b:
27          return c1
28      c3 = getchar()
29      return chr(0x10 + ord(c3) - 65)
30
31  def Keyborad_control():
32      while True:
33          global power_val
34          key=readkey()
35          if key=='up':
36              if power_val <=90:
37                  power_val += 10
38                  print("power_val:",power_val)
39          elif key=='down':
40              if power_val >=10:
41                  power_val -= 10
42                  print("power_val:",power_val)
43          if key=='go':
44              fc.forward(power_val)
45          elif key=='left':
46              fc.turn_left(power_val)
47          elif key=='back':
48              fc.backward(power_val)
49          elif key=='right':
50              fc.turn_right(power_val)
51          else:
52              fc.stop()
53          if key=='q':
54              print("quit")
55              break
56  if __name__ == '__main__':
57      Keyborad_control()
```

35〜50 行目の部分で、指示を出しているキーをそれぞれ別のキーにした。すると、コードを実行し、書き換えたもの入力したところ、ロボットカーは反応しなかった。元のキーを入力してももちろん反応しなかった。入力するキーに原因があるのだと予想し、下のようにさらに書き換えてみた。

```
1   import picar_4wd as fc
2   import sys
3   import tty
4   import termios
5   import asyncio
6
7   power_val = 50
8   key = 'status'
9   print("If␣you␣want␣to␣quit.Please␣press␣q")
10  def readchar():
11      fd = sys.stdin.fileno()
12      old_settings = termios.tcgetattr(fd)
13      try:
14          tty.setraw(sys.stdin.fileno())
15          ch = sys.stdin.read(1)
16      finally:
17          termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
18      return ch
19
20  def readkey(getchar_fn=None):
21      getchar = getchar_fn or readchar
22      c1 = getchar()
23      if ord(c1) != 0x1b:
24          return c1
25      c2 = getchar()
26      if ord(c2) != 0x5b:
27          return c1
28      c3 = getchar()
29      return chr(0x10 + ord(c3) - 65)
30
31  def Keyborad_control():
32      while True:
33          global power_val
34          key=readkey()
35          if key=='u':
36              if power_val <=90:
37                  power_val += 10
```

```
38                  print("power_val:",power_val)
39          elif key=='d':
40              if power_val >=10:
41                  power_val -= 10
42                  print("power_val:",power_val)
43          if key=='g':
44              fc.forward(power_val)
45          elif key=='l':
46              fc.turn_left(power_val)
47          elif key=='b':
48              fc.backward(power_val)
49          elif key=='r':
50              fc.turn_right(power_val)
51          else:
52              fc.stop()
53          if key=='q':
54              print("quit")
55              break
56  if __name__ == '__main__':
57      Keyborad_control()
```

指示のキーをそれぞれの頭文字を取って 1 文字のものにしてみた。すると、今度はこの書き換えたコードの通り、キーとロボットカーが反応した。実はコードの前半で、キーボード入力の設定を行う際に、getchar() という関数を用いているが、これは 1 文字しか入力できない関数なので、2 文字では反応しない。

2.5.2 障害物回避の書き換え

障害物回避のコードでも簡単な書き換えを行った。障害物を感知した際に右に回転していたのを、左に回転するように編集した。

```
1   import picar_4wd as fc
2
3   speed = 30
4
5   def main():
6       while True:
7           scan_list = fc.scan_step(35)
8           if not scan_list:
9               continue
10
11          tmp = scan_list[3:7]
12          print(tmp)
13          if tmp != [2,2,2,2]:
14              fc.turn_left(speed)
15          else:
16              fc.forward(speed)
17
18  if __name__ == "__main__":
19      try:
20          main()
21      finally:
22          fc.stop()
```

今回の編集では、14 行目の turn_right という部分を turn_left と書き換えた。すると書き換えた部分がきちんと反応し、左に曲がるようになった。この指示では、fc というモジュールを使って指示しているため、このモジュール内にある指示に書き換えたのであれば、それ通り反応するようである。

この他に、Ultrasonic センサーの設定についても編集してみたかったが、その部分はコードが複雑になっており、不可能であった。

3　Raspberry Pi の活用について

3.1　プログラミング学習をしてみて

3.1.1　準備段階について

プログラミングについての知識がなかった筆者は、OS という言葉の意味も曖昧であった。現在、一般的に使われているパソコンには、自動的に Windows や macOS が入っているため、そのような人たちも多いだろう。Raspberry Pi には、初期段階で OS が入っていないため、本体の OS を入れる作業から始めなければならない。この作業を、マニュアルを見たり、ネットで調べながら行うことで、OS とは何かということや、その入れ方について知ることができた。OS について扱うことは、コンピュータを扱うなかで、まず初めにできないといけないことなので、そこから学べるのはかなり良いと感じた。

3.1.2　Linux を扱ってみて

学習前は、Windows や macOS が普及している時代で、今更 Linux など扱う必要ないと思っていた。しかし、Linux について勉強することで、その必要性を感じることができた。

Linux はプログラミングをしたことがない人からしてみると、扱いにくく、よくわからないイメージがある。特にターミナルを開いて、そこにコマンドを用いて指示を出してコンピュータを動かすというのは、慣れるまでに時間がかかる。しかし、筆者がこの研究を行った数か月で、ある程度の操作になれるようになり、ある程度コツをつかんで動かせるようになると、エンジニアっぽいことをしている気分になり楽しかった。わからないことがあったときには、ネットで調べることで、説明の記事がたくさん出てくる。それだけの人が Linux を扱っているということだ。これは早めに扱いに慣れておくのが良いのではないかと感じた。

「1.3. Linux について」で述べたような、Linux の「オープンソース」という特徴はかなり興味深く感じた。本研究で扱ったプログラミングだけでなく、Linux 本体についてもどのような仕組みになっているのか見ることができる。今後コードについての知識を深め、コンピュータを自分なりにカスタマイズできるようになっていくともっと面白いのではないかと感じた。

3.1.3　プログラミングについて

ソースコードを実際に読むのは、やはりなかなか簡単ではない。筆者が全体の半分も解読しきれていなかったように、ソースコードの全てを解読するには相当 Python について勉強する必要がある。しかし、その勉強をする環境としては充分であると思う。一つ一つのソースコードを開いてみることができるため、それをもとに勉強すればよい。もし、Python のコードの理解に重点を置きたいのであれば、今回のロボットカー以外のキットに、日本語で、コードの説明を書いてあるものもあるので、そちらを用いるとよいだろう。また、本研究で用いたロボットカーと同じ会社の SUNFOUNDER から、スターターキットも出ている。こちらでは、Ultrasonic やモーターのようなたくさんの部品を一つ一つ動かすことができる。このキットの説明書では、部品が動いている仕組みや、動かすためのコードの説明も書いてあるため、よりエンジニアリングの専門的なことを学ぶことができるだろう。

コードの書き換えについても、コードが全部読めないうちは、難しい書き換えをするのは無理だろう。しかし、今回の研究で行ったようなとても簡単な書き換えであれば行うことができる。そして、「2.5. プログラミングの書き換え」で説明したように、書き換えに用いる vi エディタは操作がややこしく、扱いにくい。まずは

この基本的な扱い方を学ばないといけない。逆にこの基本操作を使って、簡単な書き換えを行っただけでも、その編集通りにロボットの動きが変わるというは面白いもので、プログラミングの手ごたえを少し感じることができた。

3.2 プログラミング教育への活用

実際にコンピュータのセットアップをして、プログラムを実行することで、ロボットが動くというのは面白い。コードの書き換えを行ったりする際にも、実際ロボットがどう動くのかを見ることで変化を得られるのは、よいことである。このようなキットとセットで学習することができることで、コンピュータやプログラミングについての興味関心を持たせることができるだろう。

この教材のレベルとしては、今までの学校で行ってきたパソコンの学習等に比べると、高度なことをやっており、工学系の専門的なことと比べるとかなり初心者向けである。プログラミングへの入門編と捉えることができるので、コンピュータやプログラミングに興味がある人にとっては次のステップへ進むための、ちょうどよいレベルの教材であると感じた。

近年始まったプログラム教育では、「プログラミング的思考」を育むことを目的としているため、そこにこの Raspberry Pi をそのまま教材として用いるのは難しいかもしれない。（もちろんプログラミング的思考を育むことはできるが、そこにいたるまでに手間がかかり、本来の目的を失う可能性もある。）しかし、この先 ICT がさらに普及していくにつれて、学校でのプログラミング教育ももっと専門的なことになっていく可能性がある。さらに学校のプログラミング教育に関係なく社会に出ていくためにこのような知識が必要となる時代がやってきている。それらを学ぶ教材として、この Raspberry Pi はかなり効果的であるという結論に至った。

4　まとめ

今回の研究を通して、プログラミングがいかに複雑なものなのかを感じ、習得するためには、そのための勉強をする必要があることを実感した。一方、コツをつかめばわかる部分もあり、「慣れ」というのも必要となる。どちらにせよ、プログラミングを身につけるためには、時間を要するということがわかった。そのためには、なるべく幼き頃からコンピュータになれておくのが得策である。

現在、我々の周りでは、ICT が普及している。しかし、世界標準で比較すると、日本の ICT 化はかなり遅れている。特に授業における ICT の活用について、2018 年に行われた PISA の ICT 活用調査によると、授業中のデジタル機器の使用時間は、OECD 加盟国の中で、日本が最下位であった。近年世界では、STEAM 教育という教育方法が取り入れられている。これは、Science（科学）,Technology(技術）,Engineering(工学）,Art(芸術、教養）,Mathematics（数学）の 5 つの要素を盛り込んだ教育方法である。これにより、実社会での学習に生かす能力の育成が狙いとされている。日本がこの先のさらなる ICT 化に遅れを劣らないようにするため、日本でもこの STEAM 教育をもっと積極的に取り行くべきである。その一つとして、本研究で説明したようなプログラミング学習を取り入れていくべきである。

序論で述べたように、プログラミング学習を行うために Linux 系のパソコンを新しく買おうと思えば、かなりの費用が掛かる。コロナウィルスの影響により、経済が傾いている今、「そこにお金を出すのは」とためらう人も少なくないと思う。ところが、この Raspberry Pi は本体価格が 5000 円前後で、キットや接続品も購入するにしても 2 万円前後でそろうだろう。新しくコンピュータを買うことに比べるとかなりリーズナブルな物である。コンピュータの扱いに慣れたい人や、プログラミングを始めようとしている人は、ぜひこの Raspberry Pi を用いてほしいと思う。

5　謝辞

本研究において基礎物理学研究室の高嶋教授には大変お世話になりました。研究分野に関して、全く知識のない私に、初歩的なことから優しく指導していただいたこと、厚く感謝しております。ありがとうございました。また、同じ研究室である藤野さんにも研究や発表におけるアドバイスをいただきました。この場を借りてお礼申し上げます。

論文を、TeX を用いて書く際に、本研究室の卒業生である家田さんの残していただいた「tex の使い方」という論文を参考にさせていただきました。家田さんにも感謝いたします。

6　参考文献

参考文献

[1] 国宝社　「BLUE BACKS Raspberry Pi ではじめる機械学習」　金丸隆志

[2] 国宝社　「BLUE BACKS 入門者の Python」　立山秀利

[3] 国宝社　「BLUE BACKS 入門者の Linux」　奈佐原顕郎

[4] Picar-4wd User Manual .pdf file:///D:Picar-4wd%20User%20manual.pdf

[5] github の公式ページ　https://github.com/sunfounder/picar-4wd

[6] 3 分間で人に説明できるようになる Unix と Linux の違い https://eng-entrance.com/unix_linux

[7] 世界の教育分野の ICT 利用率ランキング　https://coeteco.jp/articles/1073

[8] STEAM 教育とは　https://coeteco.jp/articles/1085