# コンピュータ理工学特別研究報告書

題目

# Raspberry Pi を用いた
# 二輪倒立振子による移動体の追跡

学生証番号　344397

氏名　河合良輔

提出日　平成 29 年 1 月 31 日

指導教員　蚊野　浩

京都産業大学
コンピュータ理工学部

# 要約

倒立振子とは,支点よりも高い位置に重心がある振り子のことである.倒立振子が倒れないようにするには,支点を重心の真下の位置にさせるような制御が必要である.その制御を行うために,本体に取り付けた二輪を利用する.その応用として,二輪移動装置のセグウェイなどが知られている.

本研究では,Raspberry Pi を用いた二輪倒立振子の制御を行う.研究は大きく分けると二段階にわけられる.まず１つ目に,Raspberry Pi を用いた二輪倒立振子での倒立制御を安定させること.２つ目に,車体に取り付けたカメラから読み取った画像を元に,移動体の追随を行えるような画像処理プログラムを作成することである.

まずは,安定した倒立制御を実現させる必要がある.そのために,本体に搭載したジャイロ・加速度センサで車体の傾きを検出しながら,車体の傾きをなくすように車輪を制御する.二輪倒立振子の制御式として,次のような運動方程式を用いる.

$u(t) = F1*\theta(t) + F2*\omega(t) + F3*v(t) + F4*d(t)$

この式において,$\theta(t)$は時刻 t における本体の傾き角度,$\omega(t)$は傾きの角速度,v(t)は車軸の速度,d(t)は車軸の移動距離,F1～F4 は,それぞれの値にかける定数である.
右辺の４つの項から求めた u(t)が,モータに加える制御量となる.右辺の４つの項を求めるために利用するのが,本体に搭載したジャイロ・加速度センサである.ジャイロ・加速度センサからは,角速度のデータを得ることができ,それを積算することで本体の傾き角度を求めることが可能になる.そうして得られたデータを利用し,F1～F4 に設定する値を試行錯誤しながら求めていくことで,Raspberry Pi を用いた二輪倒立振子を安定して倒立制御させることに成功した.

次に,画像処理と倒立振子のプログラムを同時に実行可能であることを確認する必要があり,簡単な画像処理のプログラムと倒立振子プログラムでのデータのやり取りを行いながら倒立状態を維持できることを確認した.その後,２つのプログラム間でのデータのやり取りの内容に応じて倒立振子の走行制御を行うことにした.カメラで撮影した画像に二値化処理を施し,白画素領域を求めるプログラムを作成し,求めた白画素領域の多さによって,二輪倒立振子を走行させることに成功した.走行に合わせて,移動体を判定する条件を設定することで,移動体の追跡を実現することができた.

目次

1章　序論　・・・1

2章　二輪倒立振子の従来技術　・・・2

3章　Raspberry Pi を用いた二輪倒立振子　・・・4
　3.1　二輪倒立振子の原理　・・・4
　3.2　PIC マイコンによる二輪倒立振子　・・・5
　3.3　Raspberry Pi を用いた二輪倒立振子　・・・8

4章　二輪倒立振子による移動体の追跡　・・・11
　4.1　移動体を追跡する二輪倒立振子のシステム構成　・・・11
　4.2　画像処理と倒立制御を同時に行う手法　・・・11
　4.3　画像処理の結果に応じてモータを制御する実験結果　・・・13

5章　結論　・・18

参考文献　・・・19
謝辞　・・・19
付録1 開発したプログラムの説明　・・・20
付録2 Raspberry Pi と二輪倒立振子の接続

# 1章 序論

二輪倒立振子ロボットに「セグウェイ」がある.これは,２つの車輪の間に人が乗り,体重を前にかけると前進する乗り物である.ハンドルを左右に傾けることで,回転移動を行うことができる.セグウェイで公道を走行できる国もあるが,日本では認められておらず,利用は私有地に限られる.しかし,空港や駐車場,イベント会場などの大型施設で利用されることも多くなり,セグウェイを利用することで労働効率の向上などが期待されている.現在,日本でも公道走行へ向けた取り組みは行われており,2015年7月には規制が緩和され,一定の条件を満たす全国各地域で実証実験が可能となっている.

一方,自動車の自動運転技術の実用化やドローン宅配便の試みに見られるように,従来にはないモビリティとその実用化の研究開発が進められている.これに関連して,セグウェイのようにポータブルで高い移動能力を持つ車両に自動運転の機能を搭載することが考えられる.これが実用化されれば幅広い範囲での利用が期待できる.

本研究は二輪倒立振子ロボットでの自動運転を目指している.今回の研究の範囲では,二輪倒立振子に取り付けたカメラで移動体の映像を撮影し,その画像から移動体を検出する.そして,それを自動的に追跡させることを目標とする.本研究で実際のセグウェイを使用するのは難しいので,全高20cm程度の二輪倒立振子の模型を利用する.本研究で,二輪倒立振子の製作にあたって参考にした文献を[1][2]に示す.

本論文は,次のように構成される.2章で,二輪倒立振子の従来技術に関する調査,3章で,二輪倒立振子の原理について説明する.4章では,カメラで撮影した画像から移動体を検出し,二輪倒立振子を移動体に追随するための制御について説明する.そして,5章で結論を述べる.

## 2 章　二輪倒立振子の従来技術

文献[1][2]に紹介されているように，二輪倒立振子の技術はよく知られている．ここでは，二輪倒立振子ロボットの開発事例について紹介する．

### 電動歩行アシストカー「KeePace™」

KeePace [3]（図 1）は歩行補助のための二輪倒立振子ロボットである．本体の大きさは，幅 40cm，奥行き 25cm，高さ 85cm(オレンジ)/95cm(黒)，重量は約 10kg である．KeePace のポイントとして「転倒防止」，「坂道操作」，「小回り/コンパクト」が挙げられており，それぞれについて説明する．

図 1 KeePace（https://www.youtube.com/watch?v=EbrBLitROmU から抜粋）

### 転倒防止

従来の歩行補助機器は，操作者が地面の凹凸等でつまずいた時に，機器が暴走・転倒し，操作者自身に危険が及ぶという問題があった．その問題を解決するために，KeePace は MEMS ジャイロセンサを搭載し，それを用いた倒立振子制御により，機器の転倒を防いでいる．そのアルゴリズムは次のようなものである．

(1) ジャイロセンサにより傾斜角速度を読み取り，同時に，傾斜角度を算出する．

(2) 目標傾斜角度を傾斜センサ及び目標角度発生器から得る．

(3) 目標傾斜角と(1)で計算した角度の差を角度ループ制御器で補償する．

(4) (3)で演算した角度ループ制御器の出力が目標傾斜角速度となり，(1)で読み取った傾斜角速度との差を角速度ループ制御器で補償する．

(5) (4)で演算した角速度ループ制御器の出力が主軸モータへのトルク指令となる．

このアルゴリズムを制御演算時間 1 ミリ秒で実行することで，転倒することなく，ほぼ直立な状態を維持することが可能になっている．

### 坂道操作

坂道操作では傾斜センサを利用し,上り坂に差し掛かった際に,センサで上り角度を検知することで,車体を進行方向へと傾ける.逆に,下り坂に差し掛かった際には,車体を進行方向と逆に傾けることが可能である.これにより,坂道であっても,操作者の歩くペースに合わせた安全な操作が実現できる.

### 小回り/コンパクト

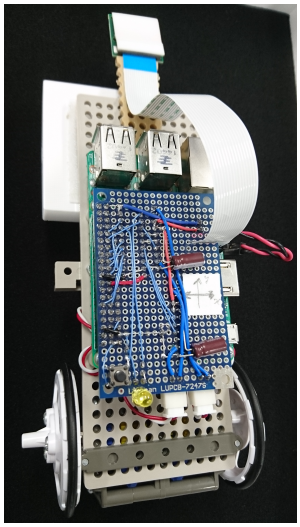
新たに独立 2 輪機構を取り入れることで,従来機器よりも旋回性能が高まり,小回り操作が可能となった.また,制御機器,駆動機構の最適設計により,車体そのものをよりコンパクトなものとした.これにより,狭い室内等での操作が容易なものとなった.

以上の点のように,KeePaceは要介護者を支援し,寝たきりや車椅子生活とならないよう,より人間らしい生活を長く送るためのアシスト機器として,活躍が期待されている.しかし,安全性や信頼性の向上,車体の軽量化など,一般の実用化に向けての課題は多く,それらの課題をクリアすることで,実際に人のアシストが可能な二輪倒立振子技術となる.

## 3 章　マイコン制御による二輪倒立振子

### 3.1　二輪倒立振子の原理

二輪倒立振子は,図 2 に示すように,２つの車輪で姿勢を安定化する装置である.本体に取り付けたジャイロ・加速度センサで角速度と加速度を測定し,傾きを補正するように２つの車輪の動きを制御する.



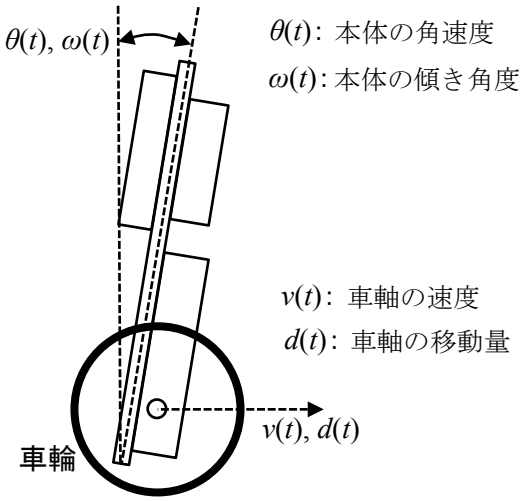



図 2 開発した二輪倒立振子の外観と制御に必要な信号

二輪倒立振子を制御する運動方程式として,式(1)を用いる.

$$u(t) = F1*\theta(t) + F2*\omega(t) + F3*v(t) + F4*d(t) \quad \cdots (1)$$

式(1)における $\theta(t)$は角速度を表す．角速度はジャイロセンサで測定する．$\omega(t)$は本体の傾き角度を表す．傾き角度は角速度 $\theta(t)$を時間積分することで推定する.$v(t)$は車軸の速度である．加速度センサの測定値を時間積分することで求めることができる.今回は,簡易的に，車輪を回転させるモータの駆動信号を積算することで推定した．$d(t)$は車軸の移動距離である．速度 $v(t)$を時間積分することで推定することができる．今回は，仮に $d(t)=0$ として研究を進めた.

これらの値を式(1)に代入し，モータに加える制御量（トルク，力）$u(t)$を決定する.F1～F4 は各項のゲインである．このような制御を十分に速い周期で実行することで，二輪倒立振子は倒立状態を維持し続ける.

図 3 に，今回の研究に利用したジャイロ・加速度センサを示す.ジャイロ・加速度センサは，MEMS 技術を用いることで小型・安価になり,モバイル機器等に広く利用されている.本研究では,InvenSense 製の I2C 接続のジャイロ・加速度センサ mpu6050 を使用した.

図 3 ジャイロ・加速度センサ　mpu6050

## 3.2　PIC マイコンによる二輪倒立振子

今回の研究の目標は,二輪倒立振子を制御しながらカメラで移動体を撮影し，画像処理を併用して移動体を追跡することである．これを実現するために，マイコンとしては処理能力が高く，ソフト開発が容易な Raspberry Pi を用いる必要がある．しかし，Raspberry Pi は厳密なリアルタイム制御を行うことができないため，Raspberry Pi で二輪倒立振子を実現することに不安要素があった．そこで，前段階として,PIC マイコンによる二輪倒立振子を製作した.

PIC マイコンは,米国マイクロチップ・テクノロジー社が製造するマイクロコントローラ製品群のことである(図4).多くの製品が存在し,組み込み機器の制御に適したアーキテクチャを有している．リアルタイム制御に適したタイマーや割り込みコントローラが一体化され，本体の大きさや消費電力が Raspberry Pi よりも遥かに小さい.

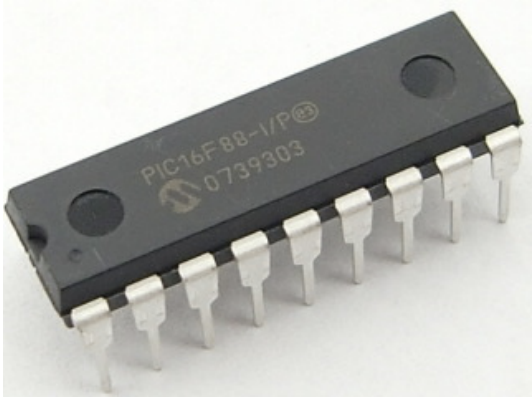



図 4 PIC マイコン

今回の開発にはPIC16F886を用いた．これは2セットのPWMに利用できるCCP1・CCP2モジュールを持っており，二輪倒立振子に取り付けた2つのDCモータを容易にPWM制御することができる．

PICマイコンの開発環境を次のように整えた．

(1) PICマイコンの統合開発環境「MPLAB X IDE」をWindows PCにインストールした．

(2) CCS社のCコンパイラ(PCM)を組み込み，C言語のプログラムを開発した．

(3) PICマイコンへのプログラムライターとしてPICKit3を使用し，書き込みアダプタとしてマルツエレックのMPIC-DPPAを使用した．

(4) デバック手段として，PICKit3 のインサーキットデバッガを使用した．これにより，PCからPICマイコンにプログラムをダウンロードすることが可能になり，ステップ実行や変数の値を確認することができる．また，PICマイコンとPCをUARTで接続することで，CCSの関数printを使用した際に，書式付き文字列がUARTに出力され，これをPC側で表示することで，プログラムの状態を確認することが可能である．

開発した二輪倒立振子の外観と回路図を図5に示す．車輪を駆動するメカは，タミヤの楽しい工作シリーズNo.168「ダブルギアボックス（左右独立4速タイプ）」をギア比38.2:1で組み立てた．これに，同シリーズNo.193「スリムタイヤセット」の55mm径のタイヤを装着し，同シリーズNo.98「ユニバーサルプレートセット」にネジ止めした．

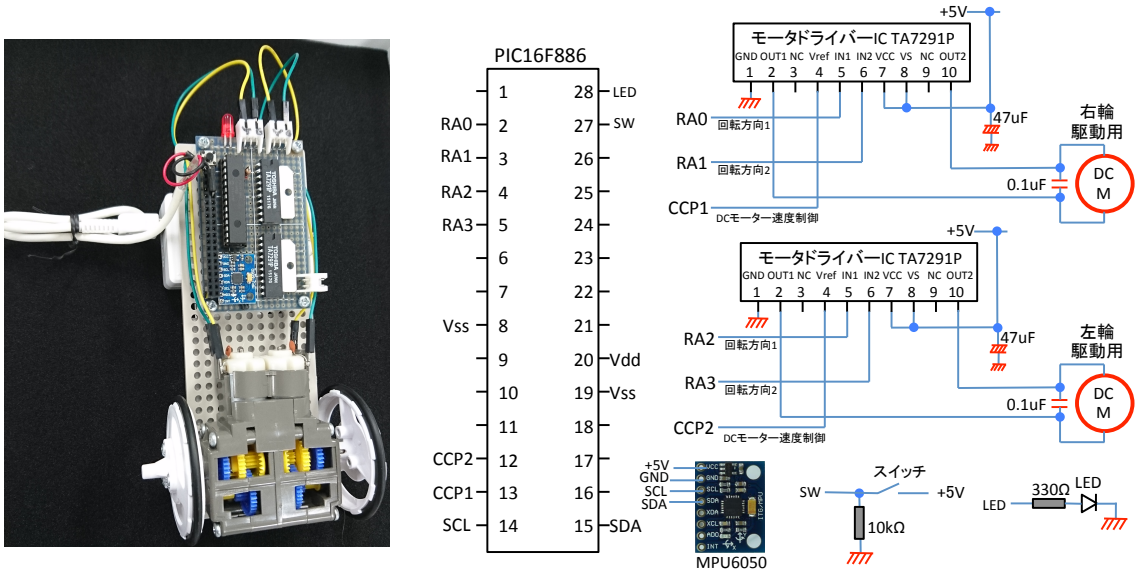



図5 PICマイコンによる二輪倒立振子の外観と回路図

PICマイコンによる二輪倒立振子のプログラムは，無限ループ処理の中で，4msec周期で，二輪倒立振子の制御を行う割り込み処理が発生し，その度にセンサから角速度の

値を読み取り,モータを 800usec 周期の PWM で動作させた.PWM は,オンとオフ(1 と 0)の繰り返しスイッチングを行い,オン時間とオフ時間の比率を変えることで，平均出力電力を制御する手法である.オン時間を周期で割った値をデューティ比という．デューティ比が高いほどモータに加える力が大きくなる.今回作製したプログラムの処理の流れを図 6 に示す.

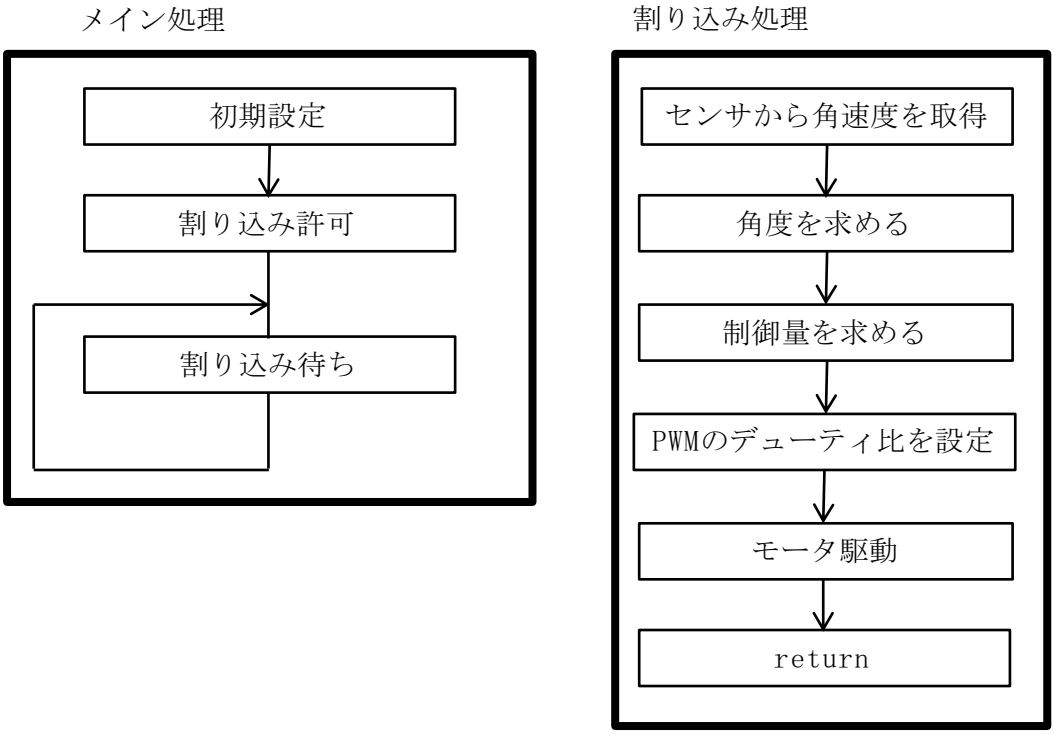



図 6　PIC マイコンによる倒立振子プログラムの流れ

二輪倒立振子の制御式は式(1) (u(t) = F1*θ(t) + F2*ω(t) + F3*v(t) + F4*d(t)) を用いる．右辺の４つの項にはそれぞれ,次のような働きがある.

F1*θ(t):車体が傾いている時，その傾きを戻す方向に車輪を駆動する．その制御量は傾き角度に比例する.ただし，この項だけでは倒立制御は安定しない.なぜならば,車体の振動により角度 0 の位置を行き過ぎる時,それを止めるための補正が働かないためである.

F2*ω(t):車体の回転運動を抑制するための項.角度 0 の位置をある程度の角速度で行き過ぎた場合,それを止めるように補正する.この項を加えると，倒立するようになる.ただし，移動を続ける.

F3*v(t):車体の移動を止めるための項．基本的に静止する.

F4*d(t):車体の位置を決めるための項．ただし，今回の実験では d(t)=0 としたので,機能していない.

F1～F4 に最適な数値を求める必要がある．解析的に求めることは難しいので，それ

ぞれの数値について,小さな値から大きな値まで様々な値を設定し,倒立振子の動きを観察しながら試行錯誤で値を決定した．最終的に決定した値と，それを変えることによる効果は次のようになった．

F1 = 1.0(値が小さいと,車輪の回転が弱く,倒立制御が働かない.値が大きいと,車輪が強く回転し,車体が振動する．どのような値を設定しても，外乱を加えると安定して倒立しないが，比較的良好な値としてF1=1.0を採用した．)

F2 = 4.0(F1と同様,値が小さいと制御が弱く,大きいと振動する．)

F3 = 50.0(値が小さいと,前後移動が大きくなる.大きすぎると制御がきかない)

F4 = 0.0(今回の実験ではうまく機能しないため,F4=0としている)

このように値を設定することで,二輪倒立振子の倒立制御に成功した.完全に静止することはないが,外部から多少の外乱を加えても,倒立状態を維持できた.開発したPICマイコンのプログラムを付録1に示す.

### 3.3 Raspberry Piを用いた二輪倒立振子

Raspberry Piは,LINUX系OSが動作するカードサイズのCPUボードであり,組み込み機器の開発に適している.ディスプレイやマウス,キーボード等を接続することにより,通常のパソコンと同じように使用することが可能である.(図7)

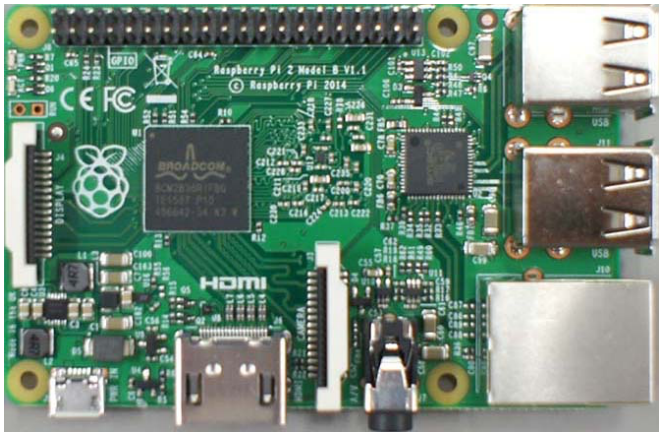
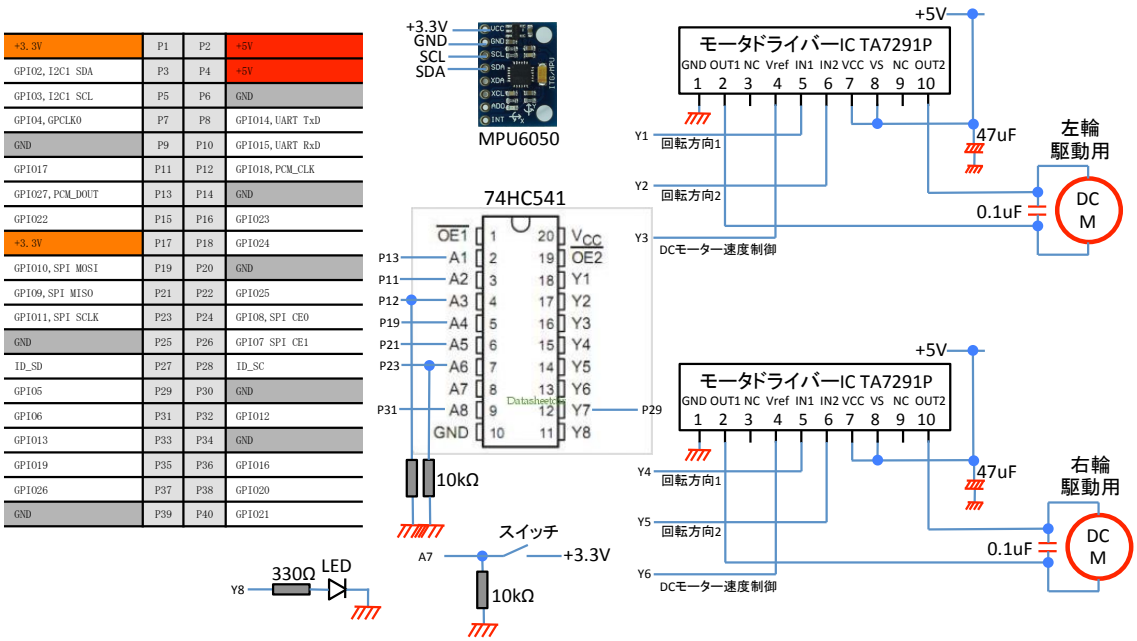



図7 Raspberry Pi 2の外観

Raspberry PiはPICマイコンと比較して処理能力は高いが，PICマイコンのように厳密なリアルタイム制御を実現することは難しい．Raspberry Piを用いた二輪倒立振子を実現することには，このような不安要素があった．そこで3.2の二輪倒立振子とほぼ同じ構成のシステムを製作し，PICマイコンの制御動作に可能な限り近いプログラムを開発することで，Raspberry Piによる二輪倒立振子の可能性を検証した．図1が作成

した Raspberry Pi を用いた二輪倒立振子の外観である．この装置には，後ほど利用するためのカメラも実装されている．図 8 に回路図を示す．２つの二輪倒立振子のハードウエア的な違いは，利用するコンピュータと電源に用いるモバイルバッテリの違いだけである．




| | | | |
|---|---|---|---|
| +3.3V | P1 | P2 | +5V |
| GPIO2, I2C1 SDA | P3 | P4 | +5V |
| GPIO3, I2C1 SCL | P5 | P6 | GND |
| GPIO4, GPCLK0 | P7 | P8 | GPIO14, UART TxD |
| GND | P9 | P10 | GPIO15, UART RxD |
| GPIO17 | P11 | P12 | GPIO18, PCM_CLK |
| GPIO27, PCM_DOUT | P13 | P14 | GND |
| GPIO22 | P15 | P16 | GPIO23 |
| +3.3V | P17 | P18 | GPIO24 |
| GPIO10, SPI MOSI | P19 | P20 | GND |
| GPIO9, SPI MISO | P21 | P22 | GPIO25 |
| GPIO11, SPI SCLK | P23 | P24 | GPIO8, SPI CE0 |
| GND | P25 | P26 | GPIO7 SPI CE1 |
| ID_SD | P27 | P28 | ID_SC |
| GPIO5 | P29 | P30 | GND |
| GPIO6 | P31 | P32 | GPIO12 |
| GPIO13 | P33 | P34 | GND |
| GPIO19 | P35 | P36 | GPIO16 |
| GPIO26 | P37 | P38 | GPIO20 |
| GND | P39 | P40 | GPIO21 |

図 8 Raspberry Pi を利用した二輪倒立振子の回路図

Raspberry Pi による二輪倒立振子のプログラムは，PIC マイコンでの二輪倒立振子のプログラムと同様，4msec ごとにセンサの値を取得し，その値をもとに制御量を導き出し，PWM を設定している．処理の流れとしては，PIC マイコンでのプログラムに近いものとなっている．

PIC マイコンでのプログラムでは，無限ループ処理の中で 4msec ごとに割り込み処理を行うという方法を取っていたが，Raspberry Pi でのプログラムでは，プログラムを遅延させる処理を挟むことで一つのループ処理の時間を調整している．

今回作製したプログラムの処理の流れを図 9 に示す．

Raspberry Pi で可能な限り正確に 4msec の制御ループを実現するため，ループ処理の開始時と終了時の時刻を調べ，その差分を求める．求めた差分の時間を 4msec から引くことで，4msec の制御ループを実現するための適切な遅延時間を求めることができる．しかし，ループにかかる時間は毎回全く同じ時間になるわけではないため，厳密には 4msec から多少の時間が前後していることがわかった．

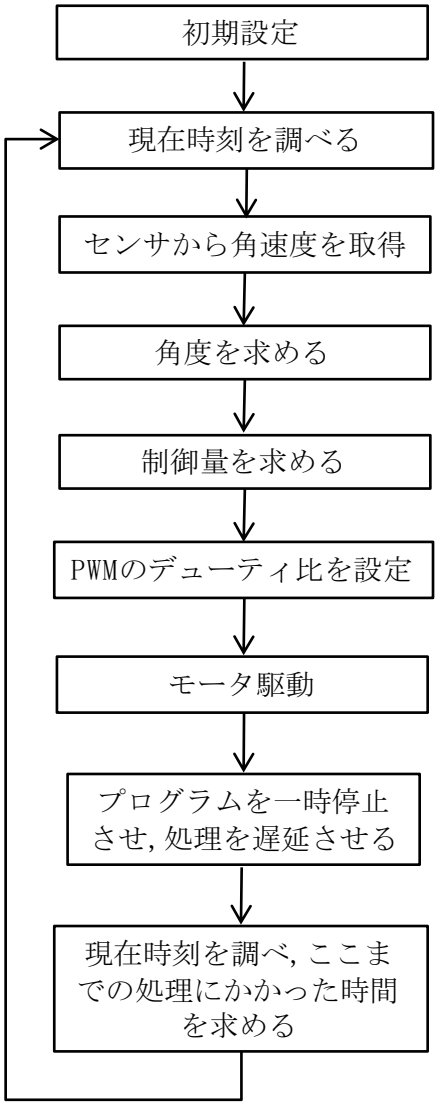



図 9　Raspberry Pi による倒立振子プログラムの流れ

3.2 と同様の方法で F1～F4 のゲインを決定した．その結果，F1～F4 の値を以下のように設定することで,倒立制御を実現できた.

F1 = 1.5(値が小さいほど,モータの回転が弱く,制御が安定しない.値が大きすぎると,必要以上の力がモータに加わり,強すぎる制御になる)

F2 = 10.0(値が小さいと制御が弱く,PIC での 4.0 程度の値では制御が弱すぎる結果となった．)

F3 = 20.0(値が小さいと制御が弱い.100.0 のような大きすぎる値だと制御が強すぎる)

F4 = 0.0(うまく動作しないため,今回の実験では 0 としている)

これらは PIC マイコンによる二輪倒立振子と，若干異なる数値になった．考えられる要因はギアボックなど部品の個体差とセットの重量の差である.

## 4 章　二輪倒立振子による移動体の追跡

### 4.1　移動体を追跡する二輪倒立振子の概要

移動体の追跡を行う Raspberry Pi による二輪倒立振子のシステム構成を図 10 に示す. 二輪倒立振子の制御だけを行う場合，ジャイロ・加速度センサから得る値を倒立振子の制御プログラムに入力し，倒立制御のために，モータの制御量を推定する．そして，その値をモータに加える.

移動体の追跡を行う場合,カメラで移動体を撮影する．その映像から画像処理で移動体を抽出し，その情報をもとに二輪倒立振子を前後・左右に移動させる.そのために,3 章で製作した Raspberry Pi による二輪倒立振子に,カメラモジュールを取り付け,移動体の映像を撮影できるようにした.これにより,図 10 に示した処理の流れで，倒立制御を行うと同時に,移動体を追跡することを可能にする.

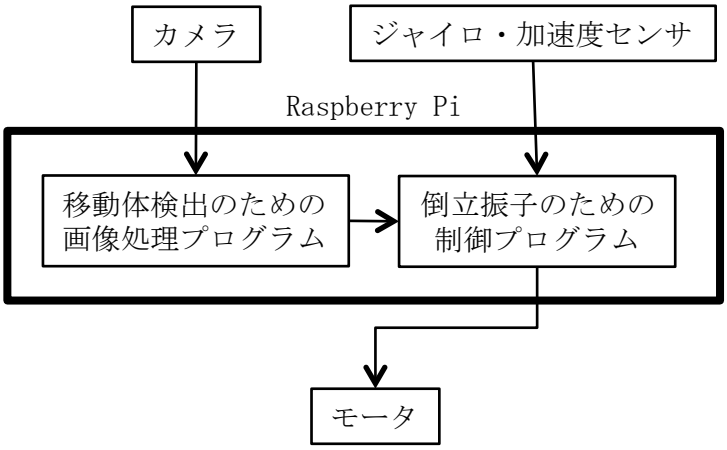



図 10 移動体を追跡する Raspberry Pi による二輪倒立振子の構成

### 4.2　画像処理と倒立制御を同時に行う手法

二輪倒立振子で移動体の追跡を行うには,画像処理プログラムによる処理結果を受け取りながら,それに応じて倒立振子の制御を行う必要がある.しかし,単純に二輪倒立振子の制御ループの中に画像処理のコードを加えると,制御ループの時間に変化が生じてしまい,それまで倒立していた状態を維持できなくなることがわかった.この問題を解決するために,制御ループの中に画像処理のコードを埋め込むのではなく，移動体の追跡に必要な処理結果が得られた場合にのみ,制御ループ側がデータを受け取るという手法をとることにした.開発したプログラムの流れを図 11 に示す.

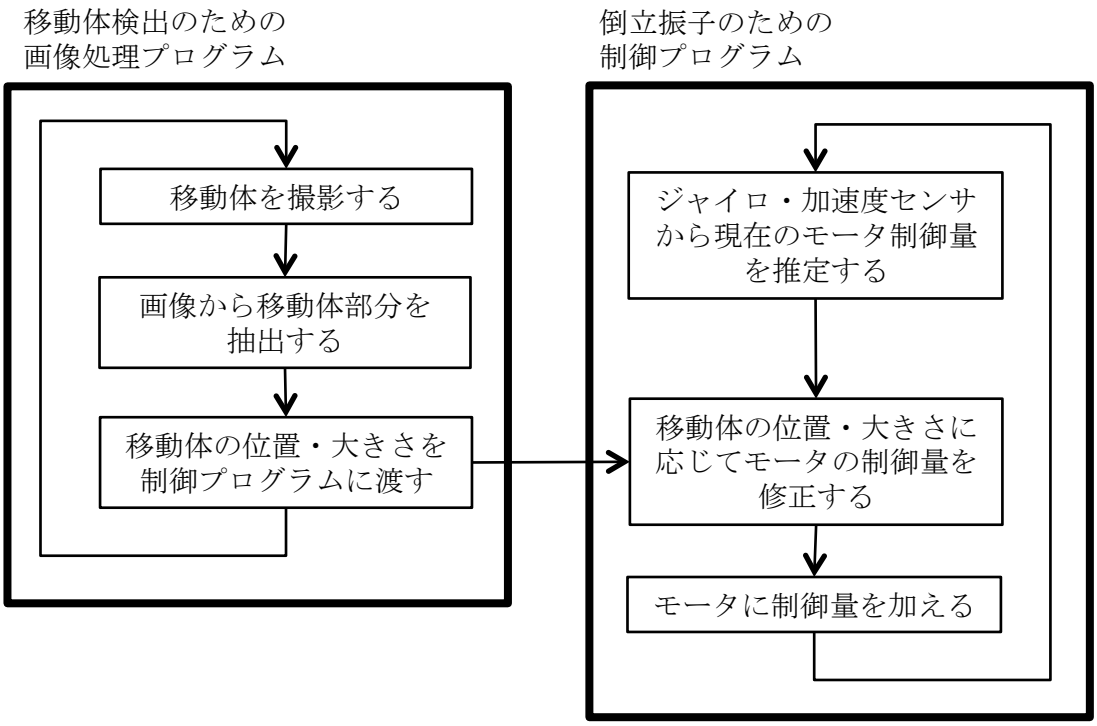



図 11 倒立制御を行いながら移動体の検出結果を受け取る処理の流れ

画像処理プログラムと倒立振子プログラム間でデータのやり取りをするために,プロセス間通信の技法の一つである「名前付きパイプ」を使った.これは,FIFO という特別なファイルを経由することでプログラム間での通信を行うものである[4].その流れを図 12 に示す. FIFO ファイルを仲介することで,画像処理プログラムから倒立振子プログラムにデータを送ることに成功した.

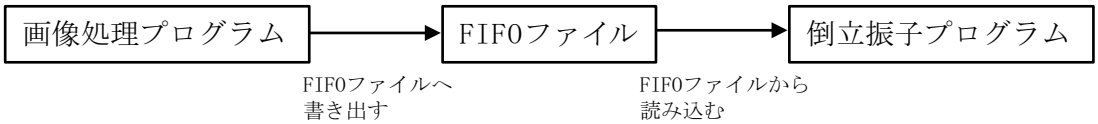



図 12 名前付きパイプによるデータのやり取りの流れ

名前付きパイプを使う手順は次のようになる. 画像処理プログラム側で FIFO ファイルを作成するために,mkfifo()関数を使用し,

“mkfifo(“ファイル名”,パーミッション)”

と記述する. FIFO ファイルを書き込み専用としてオープンするために,

“open(“ファイル名”,O_WDONLY)”

と記述する. そして,write()関数を使い,倒立振子プログラムに送りたい内容を書き出す.

倒立振子プログラム側で FIFO ファイルから読み込む場合,

“open(“ファイル名”,O_RDONLY)”
として FIFO ファイルを読み込み専用としてオープンする．そして,read()関数を使い,画像処理プログラムが送ったデータを読み込む.

以上をプログラムすることで,画像処理プログラムから倒立振子プログラムへデータを送ることが可能になった.しかし,これだけでは，FIFO ファイルが空の場合，倒立振子プログラムが画像処理プログラムからのデータが到着するのを待つことになる.これでは,倒立振子プログラムの制御ループが画像処理プログラムの処理時間に制限されるため，正しく動作しない.

FIFO ファイルの使い方について調査した結果，FIFO ファイルを読み込み専用でオープンする時に,
“open(“ファイル名”,O_RDONLY|O_NONBLOCK)”
と，NONBLOCK フラグを付けることで,FIFO ファイルにデータが無い場合,読み込み動作をしないことがわかった.これにより,FIFO ファイルが空の場合，倒立振子プログラムがデータを待つ現象をなくすことができた.その上で再度実験すると,画像処理プログラムからのデータを受け取っても倒立制御が乱れることなく,倒立状態を維持可能であることを確認した.

## 4.3　二輪倒立振子による移動体追跡の実験結果

想定したタイミングで画像処理プログラムから倒立振子プログラムにデータを転送することに成功した.次に,画像処理の結果に応じて倒立振子側のプログラムでモータを制御し,移動体の追跡を行えるようにした.

今回の実験に使う移動体として,図 13 に示すように,黒い平面上に白い紙を貼り付けたものを用意した.これは,MacBookAir の箱の裏面を使用したものである.

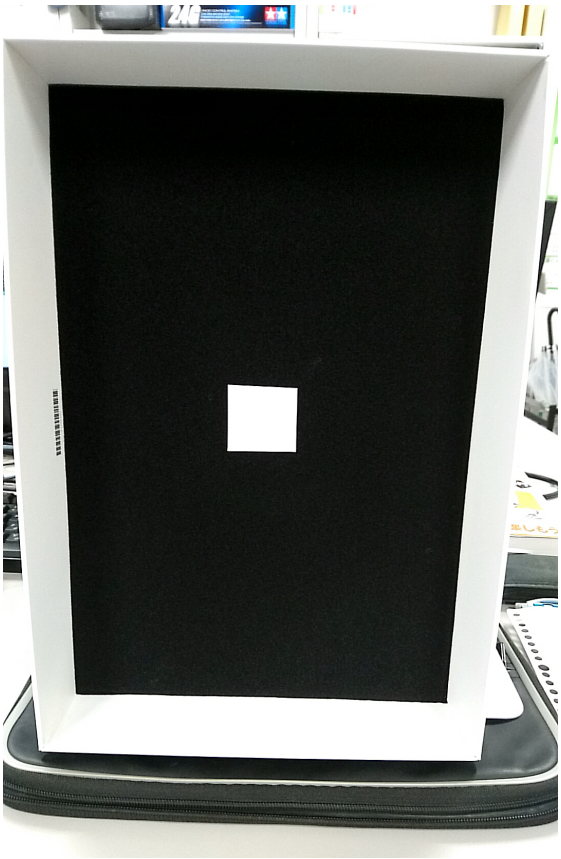


図 13 今回の実験で移動体とするもの

作成した画像処理プログラムは,カメラで撮影した画像を二値化処理した後,ROI(Region Of interest)を使い,画像の一部の範囲を指定する.ROI を使用することで,元画像から指定した範囲を新たな画像として扱うことができる.赤線で囲まれている部分が,今回 ROI で指定した範囲であり,x=170,y=90 の座標から,幅 300,高さ 300 という大きさになっている. ウインドウサイズを 640*480 に設定しているため,ウインドウ全体で 307200 画素あることになるが,今回の画像処理で必要になるのは 300*300 の範囲で,90000 画素ということになる.

指定した 300*300 の範囲がすべて黒画素である時に,範囲内に対して, 範囲内の白画素数を求めた.

指定した範囲内で求めた白画素数によって,次のように異なる文字列を書き出した.

(1)12000 より多い場合,かつ,16000 未満の場合に“停止”という文字列を書き出す.

(2)3000 以上,12000 未満の場合に“前進”という文字列を書き出す.

(2)16000 の場合に“後退”という文字列を書き出す.

倒立振子プログラムは,基本的には 3 章で作成したプログラムと同じである.そのプログラムに,条件によって異なるモータの動作を加えるというものになっている. モー

タを制御する際に,制御式の 3 つ目の項である速度を表す値を増減させることで,倒立振子を前後に走行可能なことが分かったため,今回はそのような手法で走行させる.その条件と,モータの動作は以下の通りである.

(1) “停止”という文字列を読み込んだ場合,単に倒立状態を維持し続ける.

(2) “前進”という文字列を読み込んだ場合,速度の値を増やす.

(3) “後退”という文字列を読み込んだ場合,速度の値を減らす.

このプログラムを動作させたところ,“停止”という文字列を読み込んでいる間は,多少のふらつきはあるが,その場で倒立状態を維持した.“前進”という文字列を読み込んでいる間は,少しずつ移動体に近づき,指定範囲内の白画素数を追跡する動きが確認できた.また,“後退”問いう文字列を読み込んでいる場合は,移動体から離れるよう動作することが確認できた.追跡動作のイメージを図 14 に示す.

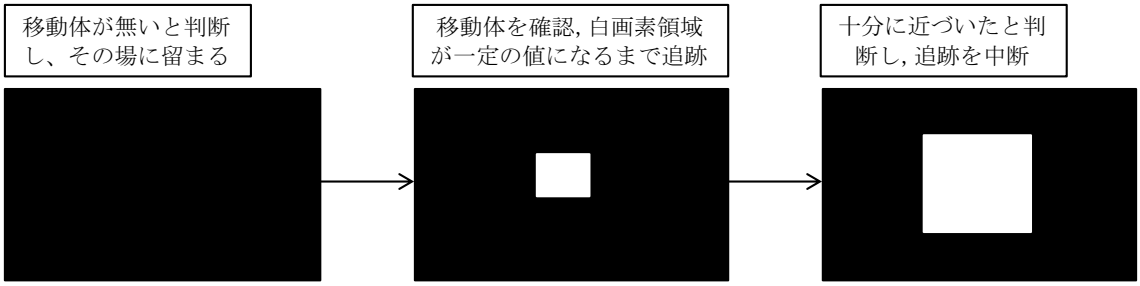



図 14 追跡動作の移動体領域の大きさ

画像処理プログラム,倒立振子プログラムそれぞれの流れを図 15,図 16 に示す.

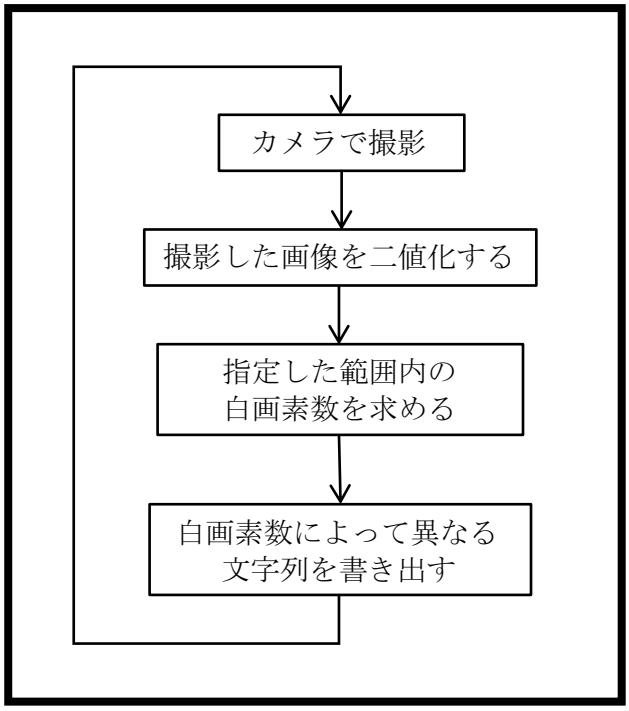



図 15 移動体追跡のための画像処理プログラム

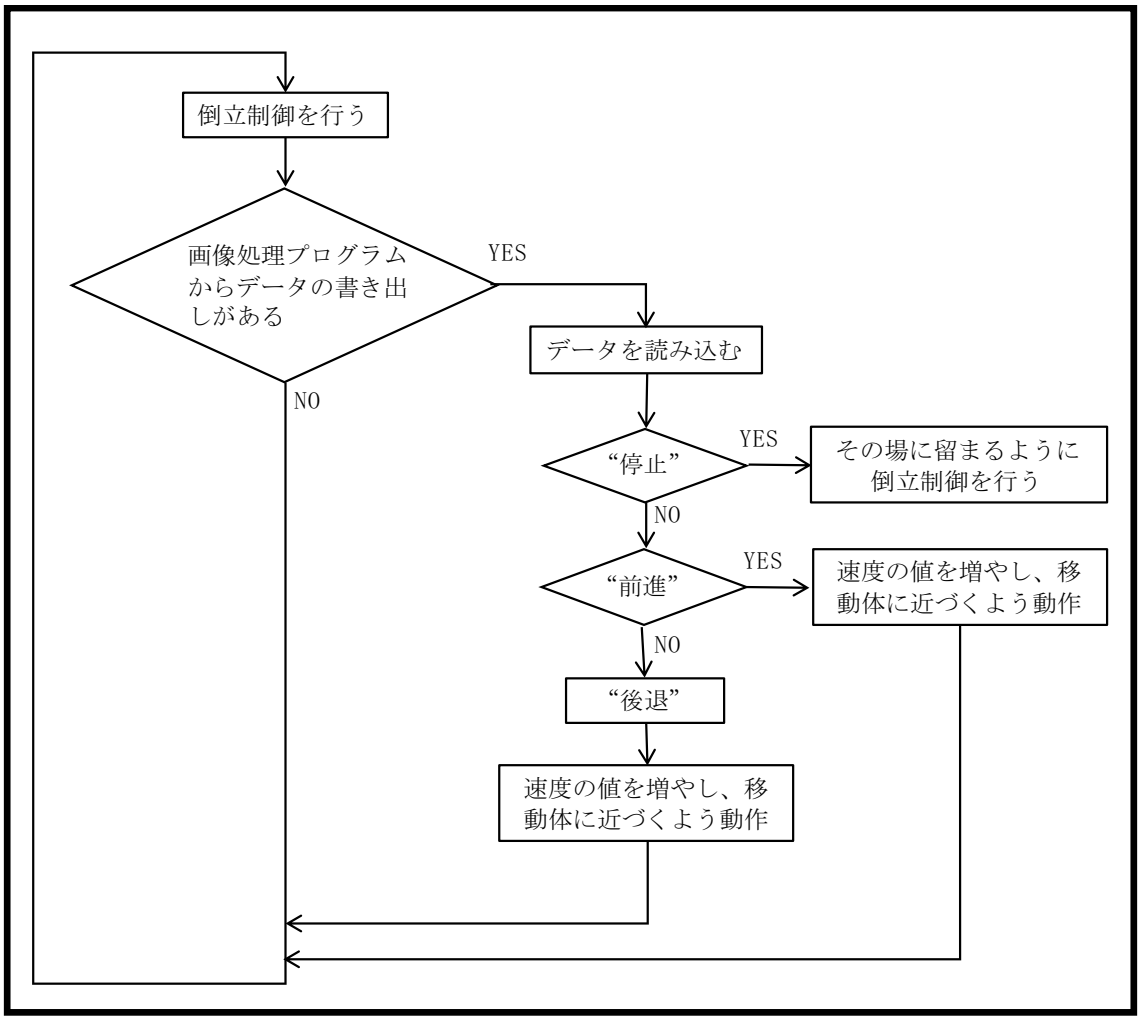



図 16 移動体追跡のための倒立振子プログラム

## 5章　結論

Raspberry Pi を用いた二輪倒立振子を製作し,Raspberry Pi にカメラモジュールを接続することで,画像を撮影できるようにした.撮影した画像を処理することで,画像における白画素領域の範囲を求め,二輪倒立振子の走行制御と連動ささせることができた.

カメラで撮影した画像を処理し,倒立制御と連動させるシステムを作ることに成功した.指定範囲内に移動体があると判断した場合に、それを追跡するような動きを実現した.移動体に十分に近づいた場合には,その時点で追跡を中止し,その場で留まるように倒立制御を続ける.

課題としては,指定した範囲内に移動体を持ってくる必要があること.カメラの画角全体から移動体だけを検出できれば,より広範囲での移動体の追跡を行える.また,今回の二輪倒立振子では,前後移動のみの走行だけで,カーブ走行は実現していない.カーブ走行が可能であれば,より柔軟な追跡が行える.

また,Raspberry Pi という安価なマイコンを使用することで,実際に販売されているセグウェイのような二輪倒立振子ロボットを使用して実験するよりも,容易に実験を進めることができた.この研究を通して,Raspberry Pi は,簡易的な実験などでの利用に便利だと考えられる.

## 参考文献

[1] 川村伸司, フィードバック制御による倒立振子ロボットの製作 Interface 2006 年 7 月号 p70~p78

[2] 半日で作る倒立振子, http://www.instructables.com/id/半日で作る倒立振子/

[3]「倒立振子制御」技術を応用した電動歩行アシストカーの開発, http://www.murata.com/ja-jp/about/newsroom/techmag/metamorphosis17/appnote/02-02, 2017 年 1 月 20 日確認.

[4] C 言語で足し算サーバープログラムの作成, http://www.geocities.jp/sugachan1973/doc/funto45.html

## 謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に感謝いたします.

# 付録 1 PIC マイコンによる二輪倒立振子のプログラム

```
#include <16f886.h>
#include <math.h>

// Pickit3 のインサーキットデバッグ機能を使う場合の CONFIG, DEBUG（NODEBUG ではなく）, MCLR（NOMCLR ではなく）
//#fuses NOPROTECT,DEBUG,NOWRT,NOCPD,NOLVP,BROWNOUT,PUT,MCLR,NOWDT,INTRC,NOIESO,NOFCMEN,BORV40

// Pickit3 のインサーキットデバッグ機能を使わない場合の CONFIG, NODEBUG, MCLR
#fuses NOPROTECT,NODEBUG,NOWRT,NOCPD,NOLVP,NOBROWNOUT,PUT,MCLR,NOWDT,INTRC,NOIESO,NOFCMEN,BORV40

//内部クロック 8MHz で利用する。
#use delay(internal = 8MHZ) //(8MHZ,4MHZ,2MHz,1MHZ,500kHz,250kHZ,125kHz,31kHz)

// I2C の定義。16F886 の場合にはハードウエアで実行する。
#use i2c(MASTER, SCL=PIN_C3, SDA=PIN_C4, FAST, FORCE_HW)

//MPU6050 とその内部レジスタのアドレス
#define MPU6050         0xD0 //0x68 を 1 ビット左シフト
#define MPU6050R        0xD1 //Read する場合、最下位ビットを 1 にする
#define POWER_MGMT      0x6B
#define CONFIG_R        0x1A
#define GYRO_CONFIG     0x1B
#define AX_ADR          0x3B
#define AY_ADR          0x3D
#define AZ_ADR          0x3F
#define GX_ADR          0x43
#define GY_ADR          0x45
#define GZ_ADR          0x47

//タイマー 0 のカウント値。例えば 4ms 周期の場合、255 - (4msec/32) = 130
#define T0COUNT         130

//関数のプロトタイプ宣言
void intTimer0();
int8 mpu6050_write(int adr, int data);
int8 mpu6050_read(int adr);
float mpu6050_rawdata(int8 reg);

#use fast_io(A)
#use fast_io(B)

//グローバル変数
float av_offset;//角速度のオフセット
float angle = 0.0f;//x 軸回りの角度
float velocity = 0.0f;//車速の推定値
float distance = 0.0f;//移動距離の推定値

#define M_PI 3.1415926f

/**** main 関数 ****/
void main() {
    int8 i;

    //ポート A の下位 4 ビットを出力、上位 4 ビットを入力に設定する。fast_io(x)を使う場合、最初に、set_tri_x()
で端子の入出力方向を決める。
    set_tris_a(0xf0);
    output_a(0x00);//0x06:順, 0x09:逆, 0x00:ストップ, 0x0f:ブレーキ
    //ポート B の下位 7 ビットを入力、B7 を出力に設定する。
    set_tris_b(0x7f);
```

```
    output_low(PIN_B7);//LED を消灯

    //MPU6050 のパワーオンでエラーが発生すれば B7 端子を 0.1sec でトグルする
    if (mpu6050_write(POWER_MGMT, 0x00) != 0) {
        while (1) {
            output_toggle(PIN_B7);
            delay_ms(100);
        }
    }

    //MPU6050 のパワーオンでエラーがない場合、LED を 3 回点滅させる。この間に、装置を平坦な場所に置く。
    for (i = 0; i < 6; i++) {
        output_toggle(PIN_B7);
        delay_ms(1000);
    }

    //平坦な場所に置いた状態で、角速度のオフセットを求める
    av_offset = 0.0f;
    for (i = 0; i < 255; i++) {
        av_offset += mpu6050_rawdata(GX_ADR);
    }
    av_offset = av_offset/255.0f;

    //角速度のオフセットを求めたことを示すために、素早く LED を 3 回点滅させる。
    for (i = 0; i < 6; i++) {
        output_toggle(PIN_B7);
        delay_ms(200);
    }

    //ccp1 と ccp2 を PWM で使う。ccp1 と ccp2 はモータの PWM 制御に利用する。
    setup_ccp1(CCP_PWM);
    setup_ccp2(CCP_PWM);

    //PWM に使うタイマー 2 を(T2_DIV_BY_16,100,1)と設定すると、PWM の周期は 0.5usec×16×100=800usec になる
    //ここで 0.5usec は 8MHz の 1 周期 0.125usec の 4 倍の値。
    setup_timer_2(T2_DIV_BY_16,100,1);
    //次の関数で設定するデューティ比の数値は、setup_timer_2()の第二引数よりも小さい数値に設定する。
    set_pwm1_duty(0);
    set_pwm2_duty(0);

    //周期的に発生する割込み用にタイマー 0 を使用する
    //タイマー 0 のクロックは 0.5us×64=32us、カウンタは 8bit。ここで 0.5usec は 8MHz の 1 周期 0.125usec の 4 倍
の値。
    setup_timer_0(RTCC_INTERNAL | RTCC_DIV_64 | RTCC_8_BIT);
    set_timer0(T0COUNT);
    //タイマー 0 による割込みを許可する
    enable_interrupts(INT_TIMER0);
    enable_interrupts(GLOBAL);

    while(1) {
        //スイッチを押すとモーターを止め、角度・速度・移動距離を 0 にする。
        if (input(PIN_B6)) {
            output_a(0x00);//0x06:順, 0x09:逆, 0x00:ストップ, 0x0f:ブレーキ
            set_pwm1_duty(0);
            set_pwm2_duty(0);
            angle = 0.0f;
            velocity = 0.0f;
            distance = 0.0f;
        }
```

```
    }
}

//タイマー０で周期的に割込みを発生させ、MPU6050 の値を読みとり、メカを制御する。
//2016 年 7 月 29 日のバージョンでは、割込み周期を 4msec に設定し、intTimer0()の処理を 2msec で終了している。
#INT_TIMER0
void intTimer0() {

    //タイマーによる割込みが発生していることを確認するために B7 をトグルする
    output_high(PIN_B7);

    //次の割込みのためにカウンタをセットする。
    set_timer0(T0COUNT);

    //角速度（度/sec 単位）を求める。
    float angular_velocity = mpu6050_rawdata(GX_ADR) - av_offset;

    //角速度を積算して角度を求める。
    angle += angular_velocity;

    //制御量を求める
#define F1 1.0f //角度に対するゲイン(1.0 に固定。第一にこれを固定して、max_pwr を調整する。)
#define F2 4.0f //角速度に対するゲイン（4.0。第二に調整する。小さいと制御が弱く、大きいと振動する。）
#define F3 50.0f //車速に対するゲイン（50。小さいと大きく前後移動。大きくすると前後幅が小さくなるが、大きすぎると破たんする。）
#define F4 0.0f //移動距離に対するゲイン。うまく動作しないので、0 に設定している。

    float f1angle    = F1*angle;
    float f2angvel   = F2*angular_velocity;
    float f3velocity = F3*velocity;
    float f4distance = F4*distance;

    float pwr = f1angle + f2angvel + f3velocity + f4distance;//制御量

    //制御量の絶対値の最大（F1=1.0 で、この値を適切に設定する。小さいと制御が遅くなり、大きいと振動する。）
    float max_pwr = 200000.0f;
    float min_pwr =  1.0f;//制御量の絶対値の最小（この値以下ではモータを駆動しない。あまり意味がない。）

    //pwm のデューティ比を設定する
    int8 duty;//pwm のデューティ比
#define DUTY_MAX 99.0f //PWM の最大デューティ比
#define DUTY_MIN 1.0f //PWM の最小デューティ比

    if ((pwr >= max_pwr)||(pwr <= -max_pwr)) {//制御量の絶対値が max_pwr を超えた場合
        duty = DUTY_MAX;
    } else if (pwr >= min_pwr) {//制御量が min_pwr?max_pwr の場合
        duty = (int8)(DUTY_MIN + (DUTY_MAX-DUTY_MIN)*pwr/max_pwr);
    } else if (pwr <= -min_pwr) {//制御量が-min_pwr?-max_pwr の場合
        duty = (int8)(DUTY_MIN + (DUTY_MAX-DUTY_MIN)*(-pwr)/max_pwr);
    } else {
        duty = 0;
    }

    if (pwr >= 0) {
        output_a(0x06);//0x06:順, 0x09:逆, 0x00:ストップ, 0x0f:ブレーキ
        set_pwm1_duty(duty);
        set_pwm2_duty(duty);
        velocity += (float)duty;
        distance += (float)velocity;
```

```
    } else {
        output_a(0x09);//0x06:順, 0x09:逆, 0x00:ストップ, 0x0f:ブレーキ
        set_pwm1_duty(duty);
        set_pwm2_duty(duty);
        velocity -= (float)duty;
        distance -= (float)velocity;
    }

    output_low(PIN_B7);
}

int8 mpu6050_write(int adr, int data) {
    int ack = 0;

    i2c_start();
    ack += i2c_write(MPU6050);
    ack += i2c_write(adr);
    ack += i2c_write(data);
    i2c_stop();

    return (ack);
}

int8 mpu6050_read(int adr) {
    int val;

    //MPU6050 と読み出しレジスタを指定する
    i2c_start();
    i2c_write(MPU6050);
    i2c_write(adr);

    //指定したレジスタから読み出し
    i2c_start();
    i2c_write(MPU6050R);//読み出しの場合には、最下位ビットを 1 にする。
    val = i2c_read(0); //0 を指定するとデータを正しく受け取るようである。
    i2c_stop();

    return val;
}

float mpu6050_rawdata(int8 reg) {
    int8 high = mpu6050_read(reg);
    int8 low  = mpu6050_read(reg+1);

    return (float)((signed int16)make16(high,low));
}
```

付録　本研究で開発したプログラムの説明などを付録とする

| | 機能 |
|---|---|
| segpi_kawai.c | 二輪倒立振子の倒立制御を行う.ファイルからある文字列を読み込んだ場合,それに応じた制御をするプログラム |
| image_kawai.cpp | カメラモジュールから得た画像を二値化し,その中の白画素領域を求め,白画素領域の大きさによって異なる文字列をファイルに書き出すプログラム |

左と右で別々のモータドライバ

同じ信号送っているが動作が違う