2019（令和元）年度　　修士学位論文

全方向移動システムの為の回路設計と制御・通信手段の開発

Circuit design and development of a control system for omnidirectional transportation

2020 年 3 月 9 日

高知工科大学大学院　工学研究科基盤工学専攻
知能機械システム工学コース

1225031　　鈴鹿　紅音

指導教員　　川原村　敏幸

目次

第1章 序論

1.1 研究背景

現代では自動車、二輪車、エスカレータ、エレベータ、動く歩道などの多種多様な移動手段が開発されおり、非常に便利な世の中になった。高速に長距離移動できるもの、一度に多くの荷物を運搬できるもの、短距離の移動を補助するものなど各々に異なる特徴がある。このように多種多様な特徴を持つ移動手段が開発されたことで移動方法の選択肢の幅が広がり、自分の目的に合った移動手段を利用することができるようになった。私たちの生活はこれらの移動手段に支えられ、今では無くてはならないものとなっている。私は、まだ世の中にない新たな移動手段を開発することで、人々がより快適な生活を送れるようになってほしいと考え本研究を行った。

1.2 移動手段と特徴

1.1 研究背景でも紹介したようにあらゆる移動手段が存在する。自動車や二輪車のように対象物が駆体と共に移動する特徴を持つものを移動型、エスカレータや動く歩道のように駆体が対象物のみを移動させる特徴を持つものを設置型として大きく 2 つに分類し、それぞれの特徴について考えた。これらの移動手段の特徴について表 1 に示す。

表 1.1　移動手段の特徴

	移動型	**設置型**
例	自動車、二輪車	エスカレータ、エレベータ、 動く歩道
利点	全方向に自在に移動可能	事故の危険性が低い
欠点	事故の危険性が高い	移動方向や速度が一定

移動型は移動方向の自由度が高いという利点が考えられる。一方で人が駆体を操縦し、危険予測や判断を行っているため、運転者の操作ミスによる事故の危険性が高いという欠点が考えられる。設置型は駆体が移動せず対象物が移動し、あらかじめ移動する軌道が分かっているため、移動型に比べて事故の危険性が低いという利点がある。一方で移動方向が一定なため、限られた方向にしか移動できないという欠点が考えられる。これらの制約に対し我々の研究室では、移動型と設置型の両方の利点を持ち合わせた、安全性が高く、全方向移動可能である理想的な移動手段の開発を目的とした研究開発を試みている。

1.3 全方向移動システムの有用性と活躍領域

移動型、設置型、人間（徒歩）の輸送の効率について考える。人間や荷物を輸送するときにかかる値段（エネルギー）のことを輸送コストとし、輸送コストと距離（速度×時間）の関係について計算し比較した。ここでは値段とエネルギーについて

徒歩で移動する場合　　：消費したエネルギー≒エネルギー補給の食費、飲料費

車に乗って移動する場合　：消費したエネルギー≒ガソリン代

のように同等として考えている。費用とエネルギーは厳密には一定ではないが、本研究ではエネルギーを生み出すために必要な値段が一定であると仮定して計算している。輸送コストを計算する計算式は以下のように示される。

$$\text{輸送コスト} = \frac{\text{消費運動量[kcal/s]}}{\text{輸送量[kg]} \times \text{速度[km/s]}}$$

人間一人あたりの輸送コストを 1 として他の移動手段の輸送コストを計算し比較すると図 1.1 のようになる。このとき自動車などの乗り物を利用する移動型の輸送コストを計算する場合は輸送量に筐体の重さを含めずに計算した。これより本移動手段を含む各移動手段の活躍領域についてグラフ化して図 1.1 に示す。

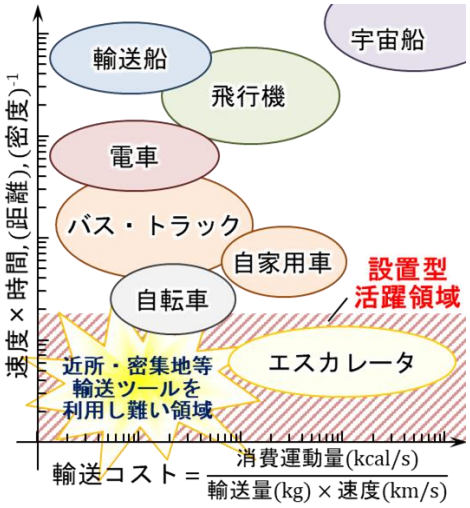


図1.1　　各移動手段の活躍領域

例えばショッピングモールや空港のような、徒歩で移動するには遠くツールを使用することが難しい密集地など、図の斜線部分に示されている領域に注目した。この領域の移動手段はエスカレータのみであり、輸送コストの安い物や全方位に対して移動可能なシステムが無く、輸送手段が未開発であると考えられる。本研究は主にこの領域を対象とした技術開発である。また移動手段だけではなく、荷物の搬送や仕分けなどの用途で使用することで輸送業界や物流業界での活躍も考えられる。

1.4 全方向移動システムの仕組み

私達が考案した全方向移動システムの仕組みについて説明する。このシステムは構成す

る六角形のユニット一つ一つに制御装置がある自律分散型輸送システムである。ユニットがハニカム形状に敷き詰められており、各ユニットの球体が自由に駆動することで、上に載っている荷物や人を搬送させるという仕組みである。ユニットを敷き詰めてシステムが完成するため、設置形状が自由に変更可能である。そのため設置範囲や設置距離、曲折した箇所など場所にとらわれずに設置ができるという特徴を持っている。敷き詰めた際のイメージを図 1.2 に示す。

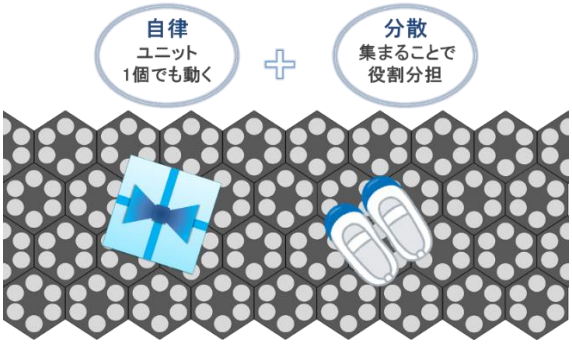


図1.2　　全方向移動システムのイメージ

1.5　システムの制御

本移動手段は移動型と設置型の両者の利点を持ち合わせた、自在に移動可能で事故の危険性が少ないという特徴があるシステムである。システム全体の制御方法は大きく分けて 2 種類考えられる。一つ目は人間（移動対象物）がシステムを制御する方法である。これは自動車のように人間側が操作、判断をする方法である。例えば、人間が目的地までのルートを決定し、操作端末などでシステムに指令を送って操作することで移動できるというイメージである。この方法は進行経路で何かが「いつ」、「どこから」、「どのような速度で来る」など様々な事象を予測する必要がある。つまり、運転者自身（人間や AI）が常に危険予測を行い、回避することを前提に移動しなければならない。この予測方式の概念図を図 1.3 に示し、「他点起発事象予測方式」と呼ぶ。

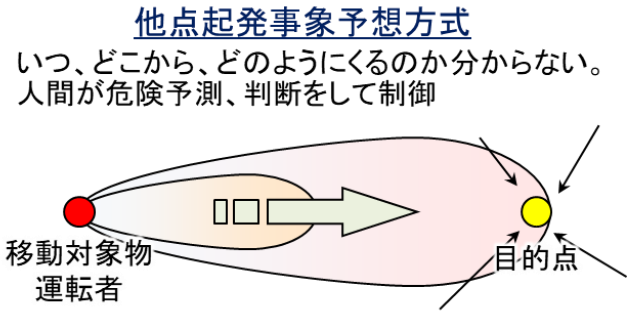


図1.3　　他点起発事象予測方式

二つ目の制御方法は、システムが人間（移動対象物）を制御する方法である。これはシステム側が人間（移動対象物）を移動させて目的地まで連れていくイメージである。この方法

は、システム自体が周囲の情報を集め、周囲の状況を把握した状態で人間（移動対象物）を移動させるため、次の瞬間にシステム上でどのようなことが起こるかを予測するだけでよい。つまり運転者自身が未来を予測する負担を自動運転システムより減らすことができる。例えば、何か進行ルートを妨げるものがあった場合に避けて通ることや、他の移動対象物の進行方向や速度を先に予測判断し、ぶつからないように制御することなどが可能となる。まさに従来とは異なる別次元の安全性を確保することが可能な手段である。この予測方式の概念図を図 1.4 に示し「自点起発事象予測方式」と呼ぶ。本システムはあらかじめ駆体が敷き詰められているため、この自点起発事象予測方式を利用できる。最終的には前者と後者の両方を取り入れた技術開発を目指す。

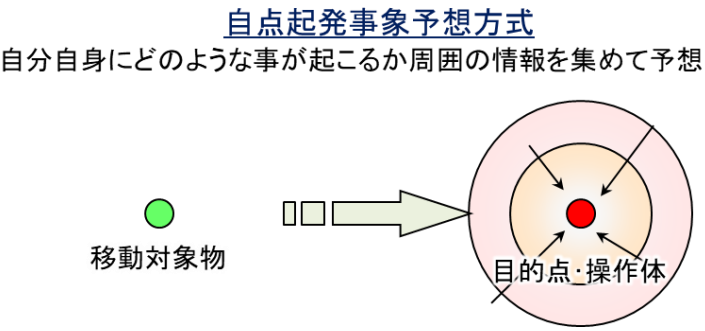


図1.4　　自点起発事象予測方式

1.6 システムの最終目標

本技術開発における最終目標は、緒言に記載したように従来にはない新しい概念の移動手段として提案することである。しかし、人間のような自由意思を持って運動している物体を移動制御することは難しいため、本研究では荷物などの静止している対象物の搬送を目標とした。そこでまずは 1 ユニットの制御を行い，次に敷き詰めて複数のユニットを制御する。その後荷物の搬送、より拡張した対象を搬送する手段と制御段階を踏んで機構開発、制御を行っていく。本論文においては 1 ユニットの制御を中心に、制御用の基本回路設計及び通信手段の構築をした。システム制御の目標についてのレベルピラミッドを図 1.5 に示す。

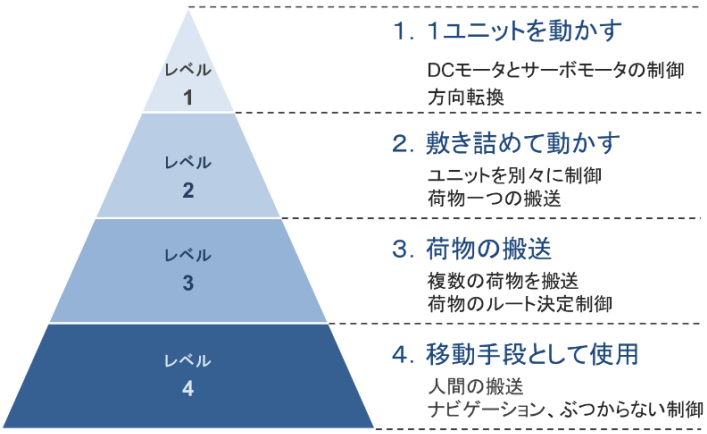


図1.5　　レベルピラミッド

第2章 先行研究

2.1 ユニットの基礎構造

先行研究ではユニットを複数個敷き詰めることでシステムを完成させるという概念を構築した。さらに本研究の基礎となる、球体を用いた三段構造を考案した。三段目に駆動用のモータがあり、その上に大きな球体が接しており、さらにその上に小さな球体を接地させ、モータからの出力を 1 つの大きな球体を介して複数個の小さな球体へ動力を伝達させることで移動対象物を搬送させる構造となっている。搬送方向はモータの出力差を利用して、2 段目の球体の回転方向を考え、1 段目の小さな球体に伝達して制御する仕組みである。ユニットの基礎構造を図 2.1 に示す。

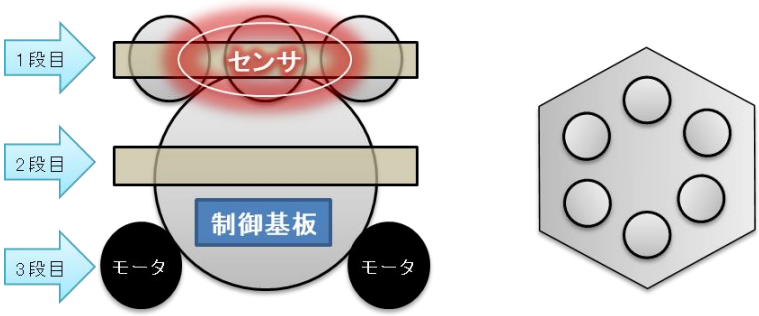


図2.1　　ユニットの基礎構造の側面図（左）と上面図（右）

ユニットの動作原理のイメージは以下のようになる。

1.センサで対象物を感知する。

2.センサ値を制御基板で受け取り処理をする。

3.モータが駆動することで下側の球体が回転する。

4.2 段目の球体の回転力を接触している上側の球体に伝達する。

5.1 段目の球体の回転力を対象物に伝達する。

本機構で三段構造にした理由について説明する。本移動手段の理想的な構造は、図 2.2 に示すように一つの球体に駆動用モータ、対象物からの信号を受け取るセンサ、制御を行うための基版が全て内蔵されていることである。

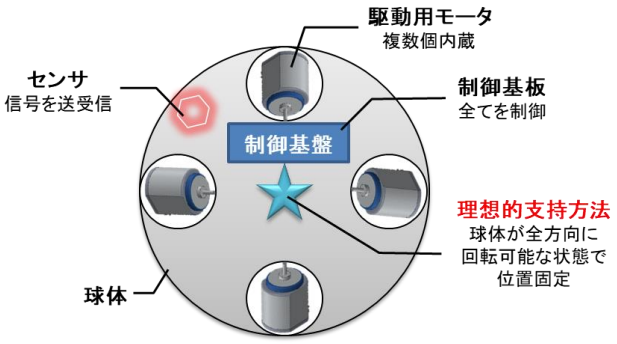


図2.2　　理想構造

しかし実際に作製することを考えると、理想構造のように一つの球体に動力源、制御基板などのすべての要素を組み込む方法が非常に困難で小型化に適していないこと、また球体が自在に動く状態で支持する理想的な支持方法が現実的ではないことが分かる。そこで他に球体を回転させる方法を検討した結果、複数の球体に対し動力源と動力伝達をする球体を接触させるというアイデアに至った。例えば 1 個の球体に対して 3 個以上のモータが必要となり現代技術でも再現可能な構造を考案し三段構造にした。

球体を用いた理由について説明する。歯車を用いて全方向に回転可能な機構を作製しようとした場合、機構や制御が複雑になるという問題点があった。そのため、歯車よりも回転方向の自由度が高い球体を用いることとした。またユニットの形状を六角形にした理由について説明する。敷き詰める球体を同じ大きさ、同じ配置で比較した場合、図 2.3 のように三角形は六角形に比べユニットの面積が大きくなってしまうことが分かる。小形の荷物や子供の足、より重い対象物を運搬できることを考えると、小型で密に敷き詰めが可能な六角形を採用した。その他本機構の由来に関しては過去文献を参考にすればよい。

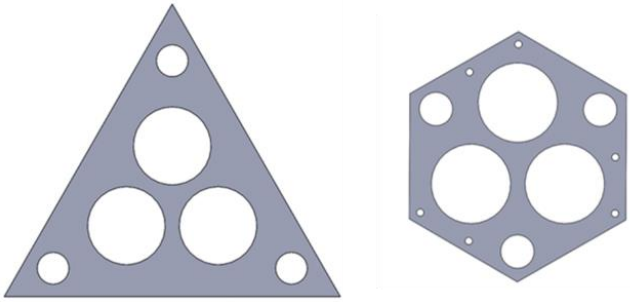


図2.3　　三角形と六角形の比較

2.2　試作機 1 号機［1］

考案した三段構造で試作機を作製した。試作機 1 号機について図 2.4 に示す。

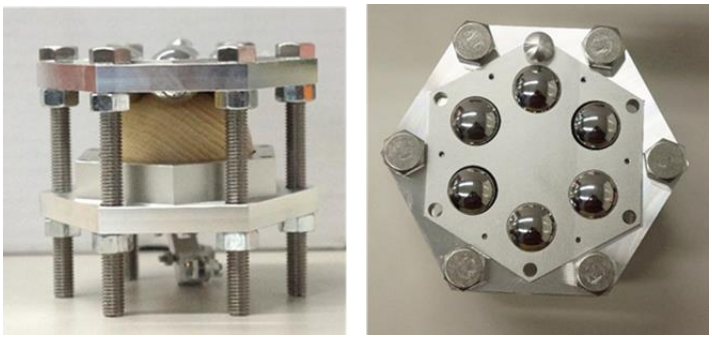


図2.4　　試作機 1 号機

試作機 1 号機では 3 段目のモータを搭載できなかった。そのため球体同士の動力伝達が可能か 2 段目の球体を手で回し動作実験を行った。実際に製作し動かしてたみたところ、問題点が明確になった。問題点は以下のとおりである。

1. ボルトを締めすぎた場合、2 段目の球体と保持器の摩擦が大きく球体が回転しない
2. ボルトを少し緩めると球体は回転するが、動力伝達がうまくいかず 1 段目に空転する

箇所がある

3. 密に敷き詰めることを考えた場合、球体のない余白部分が多い
4. ボルトの頭部分が出ているため、荷物や人が進む妨げになる
5. 駆動用モータを配置するスペースがない

これらの問題点があったため、問題点を改良した新しい構造（試作機 2 号機）を考案し作製した。

2.3 試作機 2 号機［2］［3］

試作機 2 号機について図 2.5 に示す。

図2.5　　試作機 2 号機

試作機 1 号機と比較し、形状や制御方法を大きく変更した。改良点について以下に示す。

1. 球体の数を 6 つから 3 つに減らし、直径を大きくした

試作機 1 号機では余白が多く、対象物が上に乗った時に余白部分に接触し、移動の妨げになるという問題点があった。そこで球体の数を減らし、直径を大きくすることで余白を削減した。

2. 1 段目の保持器にボールプランジャを埋め込み、保持器と球体の摩擦を軽減した

ボールプランジャは、先端にボールが付き、内部にスプリングを内蔵しているため、摺動部の位置決めや固定に適した部品である。図 2.6 にボールプランジャの詳細を示す。

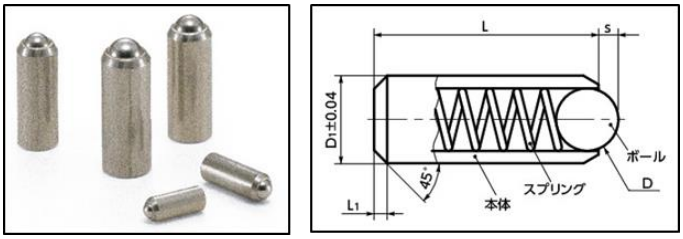


図2.6　　ボールプランジャ

試作機 1 号機は摩擦が大きく球体が回転しなかったため、ボールプランジャで球体を支持することで抵抗を減らす機構に変更した。1 段目の球体一つにつき三か所に配置したボールプランジャで固定している。詳細を図 2.7 に示す。

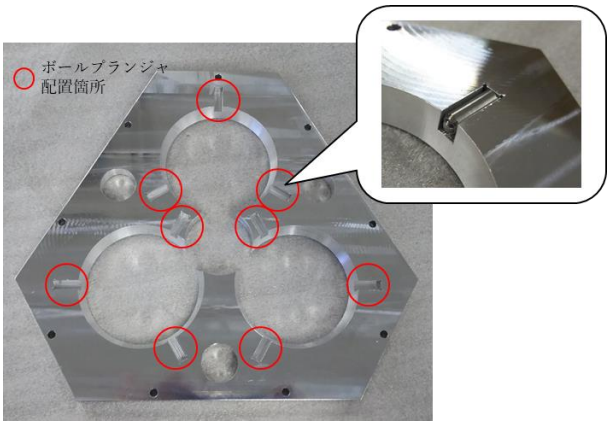


図2.7　　ボールプランジャと保持器（試作機 2 号機）

3.　2 段目の球体の保持器を無くし、軸を用いて球体を支持する機構にした

保持器の摩擦が大きく抵抗になっていることや、駆動用のモータを設置するスペースが確保できないという問題点があったため、球体に軸を通して回転させる機構に改良した。軸を通すことで 2 段目の球体の不安定さが改善され、保持器が不要になった。また球体の下のスペースにモータを配置できるようになった。詳細を図 2.8 に示す。

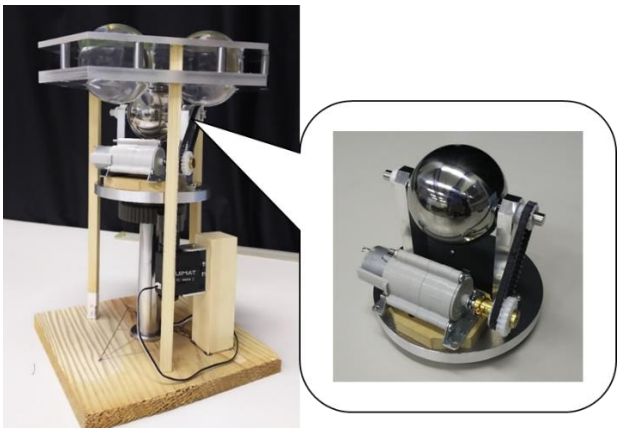


図2.8　　2 段目詳細図（試作機 2 号機）

4.　モータ数を削減し、制御パターンを簡易化した

1 号機では 1 つの球体で方位選択と搬送速度選択の両方がなされており、全方向に球体を回転させるために 3 個以上のモータを使用する予定であった。ところがこれを実現するためには、複数個のモータの速度を変更することにより移動対象物のベクトルを変更するため制御パターンが複雑になりすぎることが判明した。3 個以上のモータを用いた角度制御

のイメージを図 2.9 に示す。

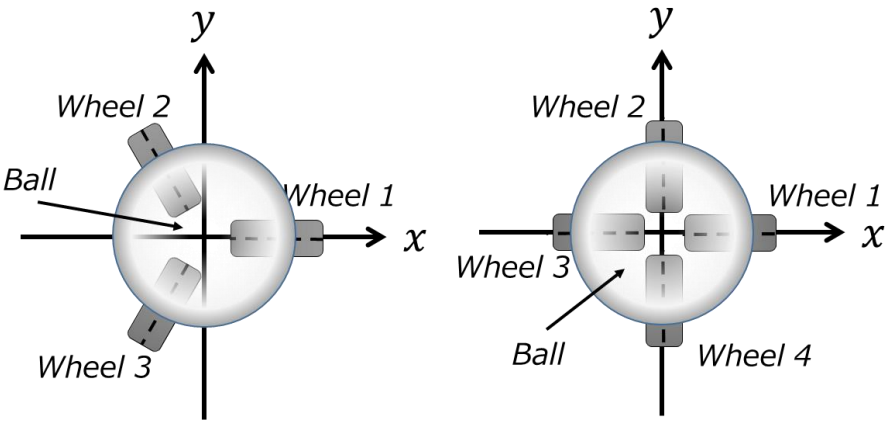


図2.9　全方向回転制御（試作機 2 号機）

そこで正転、反転しかできない搬送用のモータ（DC モータ）本体の位置を移動させ、球体と接触する位置を変更させることで球体の回転方向を制御する構造に改良した。モータの位置を移動させる構造にしたことで、搬送制御用モータと方向制御用モータの 2 種類のモータで制御ができるようになった。詳細を図 2.10 に示す。

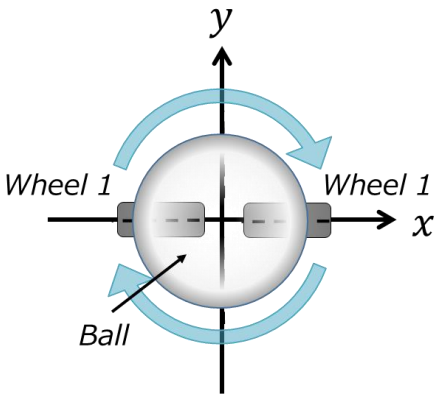


図2.10　全方向回転制御（試作機 1 号機）

5.　全方向回転制御を搬送制御と方向制御に分け、その制御に対応した機構に変更

DC モータとサーボモータなどの 2 種類のモータで方向制御と搬送制御可能な機構に改良した。テーブルの上に DC モータを設置し、ベルトを通して二段目の軸が通された球体を正転、反転、速度制御する。この部分で搬送制御を行う。三段目は正確に角度制御ができるようにサーボモータを使用した。サーボモータの回転をギアを用いて伝達し、テーブル上の DC モータの位置を変更させる。ステッピングモータではなくサーボモータを採用した理由は、サーボモータが初期位置が固定されており（常に 90 度の位置）制御が簡略化できるためである。ステッピングモータは 360 度回転可能という利点があるが、初期位置が前回に停止した位置となり、ユニットによって角度が異なってしまうことから角度制御が複雑になると考え本論文では利用を避けた。全方向回転制御機能について詳細を図 2.11 に示す。

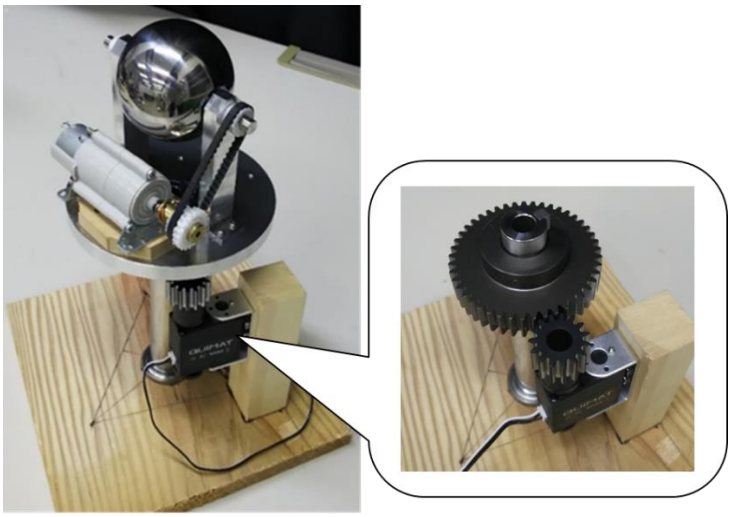


図2.11　全方向回転制御機能

2.4 試作機 2 号機の動作原理

試作機 2 号機の動作原理について説明する。動作原理のイメージについて詳細を図 2.12 に示す。

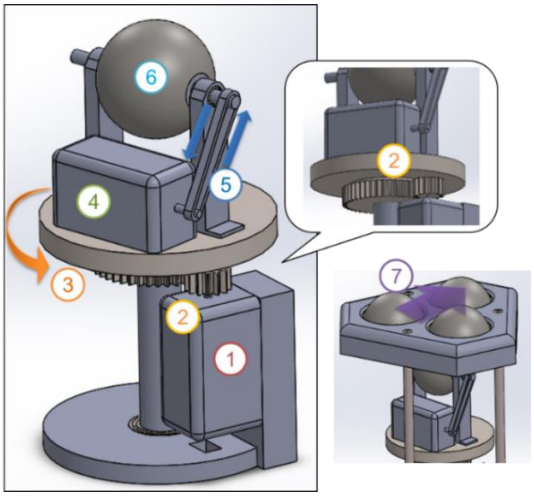


図2.12　動作原理のイメージ

方向制御
- ①サーボモータが回転
- ②モータ軸に直結した小歯車が回転し、大歯車が回転
- ③大歯車と共にテーブルが回転し移動方向決定

搬送制御
- ④テーブル上の DC モータが回転
- ⑤DC モータの回転をベルトで軸に伝達
- ⑥球体が回転
- ⑦一番上の 3 個の球体が回転
- ⑧上に乗っている荷物や人が搬送される

2.5 試作機 2 号機の問題点

実際に作製し動作実験を行ったところ、また新たな問題点が見つかった。問題点について説明する。

1. ユニットのサイズが大きすぎる

ユニットのサイズが大きく球体も大きくなってしまったため、敷き詰めた場合に搬送物との接触面積が低くなってしまう問題点が見つかった。足が小さな子どもや小さな荷物などの搬送が難しくなるため、ユニットと 1 段目の球体の小型化が必要であると考えられる。

2. 敷き詰めの際に別のユニットが必要となる

3 個の球体を用いて、球体のない余白部分をより少なくすることを考えた結果、1 段目の形状が三角形に近い六角形の形状になった。しかし正六角形ではないため、敷き詰める際に別の形状のユニットが必要となる問題が発生することが分かった。サイズや形状の異なるユニットも必要になると、コストや部品数、手間が増えてしまうため試作機 1 号機のような正六角形が好ましい形状であると考えられる。参考までに 2 号機を構想した際の装置を敷き詰めたイメージを図 2.13 に示す。

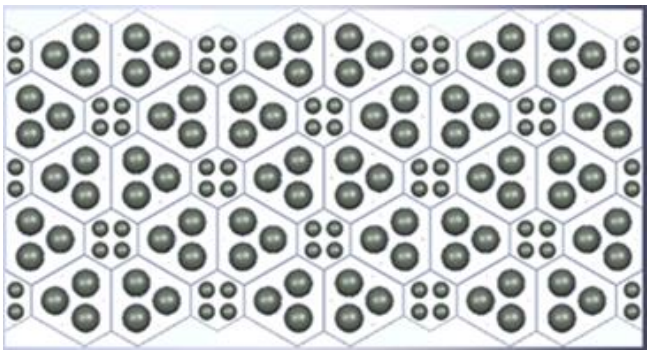


図2.13　敷き詰めイメージ

3. 1 段目の球体がうまく回転しない

動作実験を行った結果、1 段目の球体が滑っていることと球体が上下にはねてしまうということが分かった。素材と保持方法の検討が必要である。

第3章 試作機 3 号機および 4 号機の作製

3.1 試作機 3 号機

試作機 3 号機では 2 号機の問題点を改良し更に構造の最適化を行った。試作機 3 号機について図 3.1 に示す。

図3.1　　試作機 3 号機

3.2 試作機 3 号機の改良点

試作機 2 号機から改良した点について説明する。

1.　ユニットの小型化と全体の形状を変更

ユニットのサイズが大きすぎたために全体を小型化した。また 2 号機試作時に見られた敷き詰める際に別ユニットが必要となる問題点は、全体を正六角柱の領域内に収めるような機構に変更することで補った。2 号機開発時のポイントとなった余白の多さは、ボールローラーという部品を用いることで補った。ボールローラーを図 3.2 に、ボールローラーを使用した 1 段目の詳細を図 3.3 に示す。

図3.2　　ボールローラー

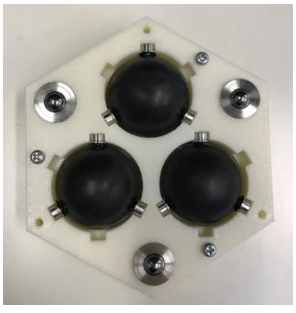


図3.3　　1 段目詳細（試作機 3 号機）

2.　球体の素材と保持方法を変更

下側の金属球と 1 段目のアクリル球との摩擦が小さかったことが原因で滑りが発生したのではないかと考察し、滑りを防ぐために球体の素材をアクリルからゴムに変更した。また試作機 2 号機ではボールブランジャを保持器の下側のみに組み込み、ボールを保持していたが、球体が跳ねてしまうという問題が生じていた。そこでボールローラーを上下の保持器に組み込み、球体を挟み込むことで跳ねを抑える構造に変更した。このように改良することで跳ねを抑え、さらに球体と保持器との間の摩擦をも軽減できるものと考えている。図 3.4 に試作機 2 号機と試作機 3 号機の球体保持方法の比較を示す。

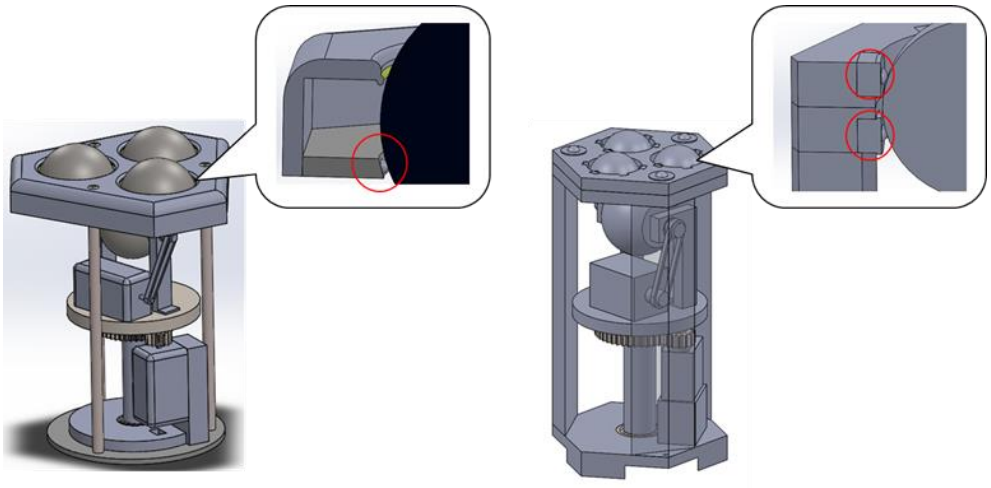


図3.4　　試作機 2 号機（左）と試作機 3 号機（右）の球体保持方法の比較

3.　球軸を支える柱の形を変更

ユニットの小型化により、DC モータの設置スペースが確保できなくなった。モータのサイズを小さくすると搬送トルクも小さくなってしまうため、試作機 3 号機では逆 L 字のような形に変更し、同じ DC モータを設置できるようにスペースを確保した。図 3.5 に試作機 2 号機と試作機 3 号機の変更した柱の比較を示す。

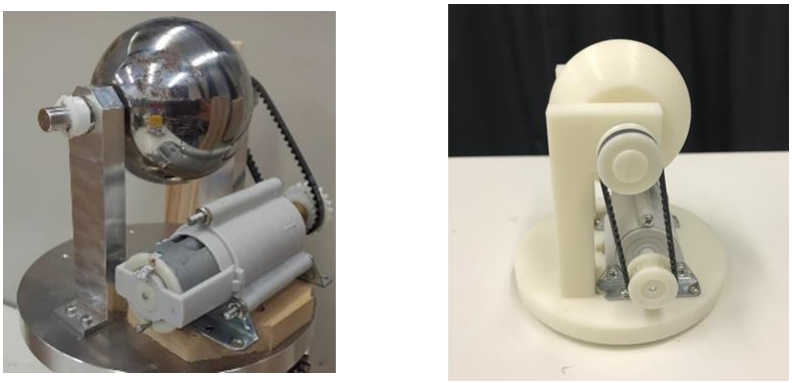


図3.5　　試作機 2 号機(左)と試作機 3 号機（右）の変更した柱の比較

3.3 試作機 4 号機

試作した 4 号機を図 3.6 に示す。

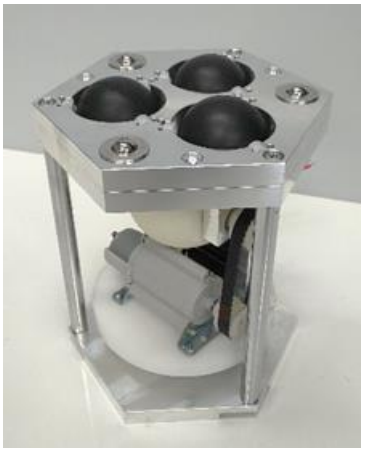


図3.6　　試作機 4 号機

試作機 3 号機ではユニットの小型化など試作機 2 号機から大幅な変更を行ったが、歪み等の影響を減らすために高さ方向も縮小できないかと思案した。そこで試作機 2 号機ではテーブルの下にあったサーボモータをテーブルの上に設置し、テーブルと一体になったサーボモータがまわることで回転方向を決定する構造に変更した。このように変更したことで高さが約 24cm から約 15cm になり、約 9cm も小型化することができた。図 3.7 に試作機 3 号機と試作機 4 号機の変更した部分の比較を示す。

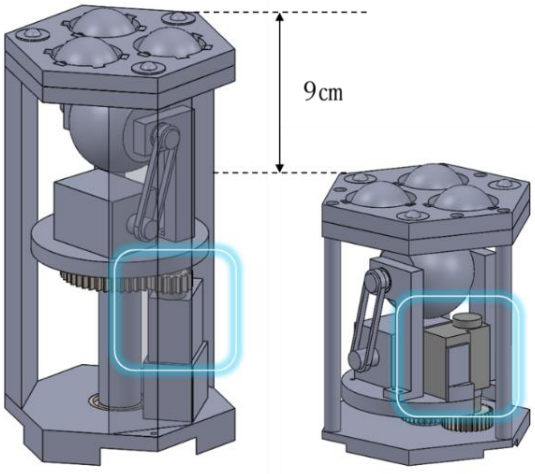


図3.7　　試作機 3 号機（左）と試作機 4 号機（右）のモータ位置の比較

本研究ではこの 3 号機、4 号機の制御機構について回路設計およびプログラム開発などに取り組んだ。

第4章DC モータとサーボモータの制御

4.1 モータの制御目標

最初は 1 ユニットを動かすことを目標にして、モータの制御を行った。ユニットは DC モータで搬送、サーボモータで方位決定をする仕組みになっている。そこで DC モータの制御、サーボモータの制御、DC モータとサーボモータの同時制御の順番に制御を行った。DC モータは正転、反転、速度変更の制御ができる事、サーボモータは滑らかに角度変更ができる事を目標とした。制御を行った順番に詳しく説明する。

4.2 DC モータの制御

1 ユニットに搭載されている DC モータは 1 個だが、制御練習とプログラミング習得をするために 2 個のモータをボリューム抵抗で制御した。1 個のボリューム抵抗で両方のモータの速度を制御し、残りのボリューム抵抗で左右のモータの速度変更を行った。速度の違いが確認しやすいように LCD を取り付け、値を表示している。モータの制御の様子を図 4.1 に示す。

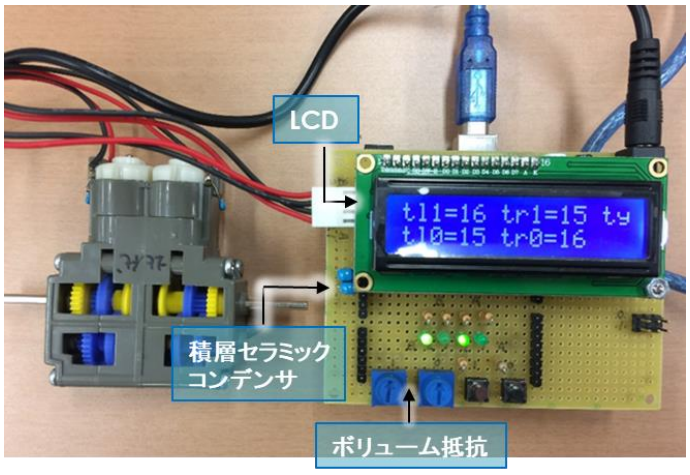
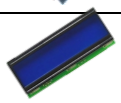


図4.1　　DC モータ制御用回路

この回路を作成する際に使用した部品の仕様を表 4.1 に示す。

表4.1　　DC モータ制御用回路の部品

名称	用途、役割
ArduinoUNO 	マイコン
積層セラミックコンデンサ 	ノイズ（交流電流）を逃がす
USB シリアル変換モジュール 	パソコンと接続して データの送受信をする
LCD モジュール 	液晶ディスプレイ
ボリューム抵抗（可変抵抗器） 	つまみをまわすことで抵抗値を変えられる抵抗器

マイコンは ArduinoUNO を使用した。マイコンからの出力は基本的に ON か OFF のどちらかとし、電圧の強さを変えて制御を行った。電圧の上昇や下降による無駄な熱が発生することで効率が悪くなると推測し、ON、OFF の時間の比率（デューティ比）を変えて回転数を変化させる PWM 制御を用いた。作成したプログラムを読み込ませて動かしたところ、全体の速度制御と左右の速度変更、正転反転制御のどちらも正確に動作することを確認できた。

4.3　サーボモータの制御

サーボモータを制御するための回路を自作した。自作した回路を図 5.2 に示す。

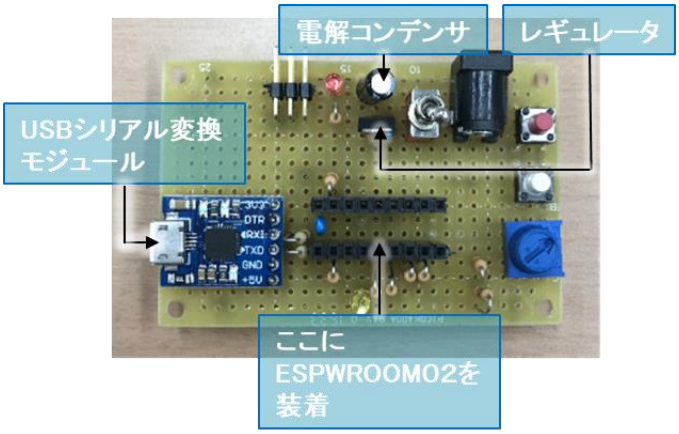
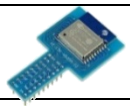
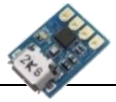
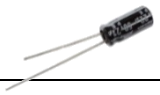


図4.2　　サーボモータ制御用回路

この回路を作成する際に使用した部品の仕様を表 5.2 に示す。

表4.2　　サーボモータの制御回路に使用した部品

名称	用途、役割
ESP-WROOM02 	無線通信用モジュール
USB シリアル変換モジュール 	パソコンと接続して データの送受信をする
積層セラミックコンデンサ 	ノイズ（交流電流）を逃がす
固定正出力 ロードロップアウトレギュレータ 	消費電力を低く抑え、 入出力間の電圧差を小さくする
電解コンデンサ 	電圧を安定させる
ボリューム抵抗（可変抵抗器） 	つまみをまわすことで抵抗値を変えられる抵抗器

今回は USB シリアル変換モジュールを搭載しデータの送受信を行った。また今回は使用しないが将来的に遠隔操作ができるようにしたいと考え、無線通信用モジュール（ESP-WROOM02）も搭載した。プログラムを作成し、DC モータの制御と同じようにボリューム抵抗で角度変更をして動作実験を行ったところ、問題なく動いた。

4.4　DC モータとサーボモータの同時制御

サーボモータを制御するための回路を自作した。自作した回路を図 5.3 に示す。

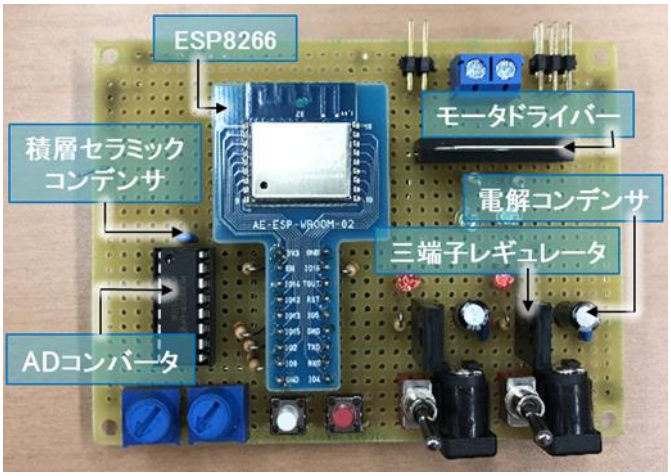
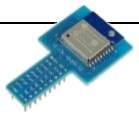
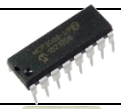
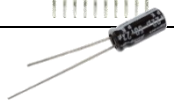
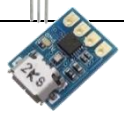


図4.3　　同時制御用の回路

この回路を作成する際に使用した部品の仕様を表 5.3 に示す。

表4.3　　同時制御の回路に使用した部品

名称	用途、役割
ESP-WROOM02 	無線通信用モジュール
積層セラミックコンデンサ 	ノイズ（交流電流）を逃がす
AD コンバータ MCP3008 	アナログ信号をデジタル信号に変換
モータドライバ TA7291P 	モータの制御用集積回路
電解コンデンサ 	電圧を安定させる
三端子レギュレータ TA48033S 	入力電圧を変換し、 出力端子側に必要な電圧を発生させる
USB シリアル変換モジュール 	パソコンと接続して データの送受信をする
ボリューム抵抗（可変抵抗器） 	つまみをまわすことで抵抗値を変えられる 抵抗器

DC モータの制御とサーボモータの制御が 1 つの基板で同時にできるように回路を作製したため、実際に同時制御可能か動作実験を行った。今回もボリューム抵抗で速度制御と角度制御を行った。動作確認用のプログラムを作成し、動作実験を行ったところ DC モータの速度変更とサーボモータの角度変更ができた。

第5章 通信制御（無線通信）

5.1 無線通信の種類

次にどのような通信手段で複数のユニットを同時に制御するかに関して構想した。通信方法は有線通信と無線通信が考えられる。膨大な数のユニットを敷き詰めることも考えると、無線通信のユニット間の配線が要らないという利点が適しているのではないかと思われる。配線する場所やケーブルの収納場所に悩む必要が無くなり、敷き詰めた後のメンテナンスも容易になると想定している。またユニットの個数を増やす場合や、壊れたユニットを一部分だけ取り換える時に配線を変えなくてもよいというメリットも考えられる。このようにユニットの個数や配置を自由に操作しやすいという点や遠隔操作ができる点から本研究では無線通信を選択した。無線通信には幅広く種類があったため、日常生活でもよく使われている赤外線、Bluetooth、Wi-Fi の三種類を比較してみることにした。3 種類の無線通信について表 5.1 に示す［4］。

表5.1　無線通信の特徴

	Wi-Fi	Bluetooth	赤外線通信
利用方法	インターネット通信などインターネット通信には無線 LAN アクセスポイントを介す	PC とマウス・キーボード、コードレスイヤホンなど機器同士で常時接続するものなど	携帯電話同士のデータ交換、家庭用リモコンなど機器同士で随時接続するものなど
特徴	・電波の届く範囲であればどこでもネット接続可能 ・電波が届きにくい場所では接続不良になることがある	・途中に障害物があっても電波が届く ・電子レンジや無線 LAN などと使うと電波干渉を起こす	・コストが低い ・送受信両者が近距離で妨害するものがない状態でしか通信ができない
通信距離	約 25～50m （屋内）	半径 10m～100m	30cm～1m
通信速度	最大 54Mbps	最大 24Mbps (規格による)	最大 16Mbps (規格による)

表 5.1 に示した 3 種類の通信の中で最も通信速度が速く、遠距離通信ができる Wi-Fi を使用することにした。

5.2 Web サーバの実装

無線通信での第一制御目標は、1 ユニットの DC モータとサーボモータを遠隔操作で制御することである。1 ユニットの制御が確認できた後、複数のユニットを遠隔操作で個別に制御することを目指す。回路は同時制御用の回路と同じものを使用した。制御方法は、最初に ESP8266（無線通信用モジュール）に Web サーバを実装させる。次に Web クライアント（PC、タブレット、スマートフォンなど）を Wi-Fi ルータに接続する。次に Web サーバを実装させた ESP8266（無線通信用モジュール）を同じ Wi-Fi ルータ（アクセスポイント）に接続する。接続が成功していることを確認出来たら、Web クライアントから作成した Web ページにアクセスし、画面に表示されたスイッチで制御を行う。制御イメージを図 5.1 に示す。

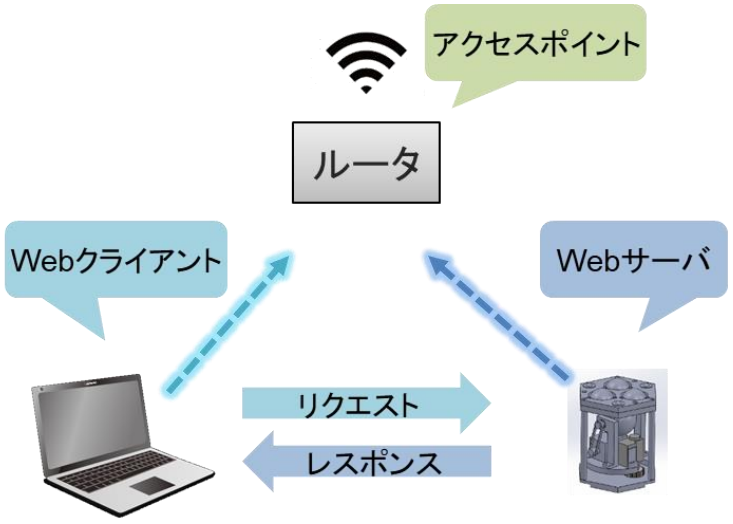


図5.1　Web サーバ実装制御のイメージ

動作確認を行った結果、ユニット 1 つ分の制御が確認できた。しかしユニットを増やして制御することを考えると、増やすユニット数分の Web クライアントが必要となること、制御するユニット数分の Web ページを立ち上げる必要がある事が判明した。Web ページを大量に開くことでプログラム量が膨大になり、通信速度も遅くなることが予測できる。1 ユニットにつき PC やタブレットが必要になりコストがかかることや、通信速度が落ちて動作が重くなることを考えると、この制御方法は多ユニットの制御に適していないといえる。そこで複数のユニット制御ができる方法を考索し別の制御方法に移行した。

5.3 TCP 通信と UDP 通信［5］［6］

Web サーバ実装制御が多ユニットの制御に向いていないという問題点があったため、その他の通信方法で制御を行うことにした。そこで TCP 通信と UDP 通信に着目した。TCP 通信とは Transmission Control Protocol 通信の略で通信プロトコル(通信規約)のひとつである。送信側が受信側にデータを送信した後、送信側と受信側がデータの送受信ができているかどうかを相互に確認する。もし確認できない場合はもう一度データを送信するという通信である。例えば電子メール、ファイル転送、www、HTTP など転送効率より確実性が重視される用途でよく利用される。このようにお互いに確認をして送受信する通信であるため、信頼性は高いが転送速度が低いという特徴がある。TCP 通信のイメージを図 5.2 に示す。

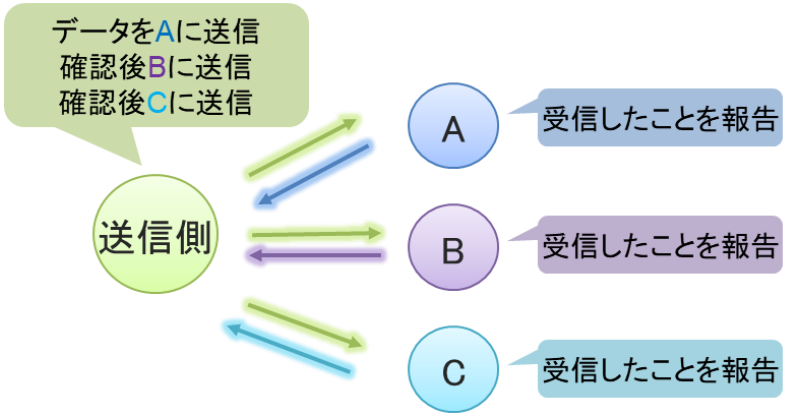


図5.2　TCP 通信のイメージ

UDP 通信とは User Datagram Protocol 通信の略で通信プロトコル（通信規約）のひとつである。送信側が受信側にデータを一斉送信し、受信側が自分宛てのデータのみ受信するという通信方法である。受信側の確認をせずに一方的にデータを送り付け、スピードは速いが情報が洩れず正確に届いているかは分からない通信である。例えば音楽のストリーミング配信やプリンタに対して印刷データを送るときなど、人間が認識できないほどの微量のデータが欠けるよりも大量のデータが速やかに相手に届く方が重要というときに使われる。このようにスピード重視な通信方法であり、重いデータやたくさんのデータの処理に向いているという特徴がある。UDP 通信のイメージを図 5.3 に示す。

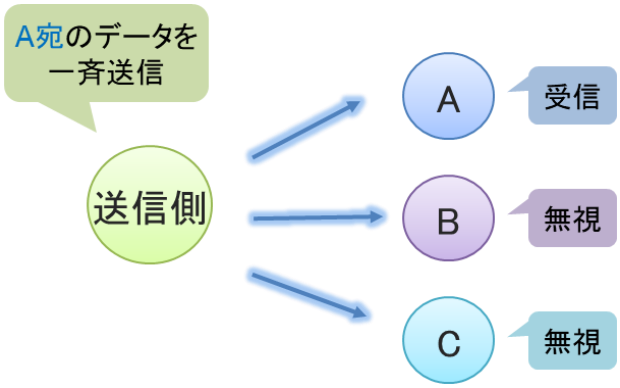


図5.3　UDP 通信のイメージ

TCP 通信と UDP 通信を比較すると、制御をするユニット数が少ない場合は信頼性の高い TCP 通信が適していると思われる。一方で UDP 通信は一対多の通信で高速でリアルタイム性がある特徴が多ユニットの制御に適しているのではないかと考えた。将来的に多数のユニット制御をすることを考慮し、本研究では UDP 通信を用いて制御を行うことにした。

5.4 UDP 通信を用いた制御

制御目標は、パソコンと回路（ユニット）間でデータの送受信を行い、パソコンで操作した通りにユニットが動く状態にすることである。制御方法は、最初に Web クライアント（PC、タブレット、スマートフォンなど）を Wi-Fi ルータに接続する。接続された Web クライアントから、全てのモータの速度や角度のデータを UDP 通信で回路（ユニット）に一斉送信する。一斉送信されたデータを回路が受信し、自分宛てのデータのみ処理することでそれぞれのユニットの速度や方位が変更されるという方法である。このとき IP アドレスとポート番号でアドレス割り振りすることでそれぞれのユニットを識別している。制御イメージを図 5.4 に示す。

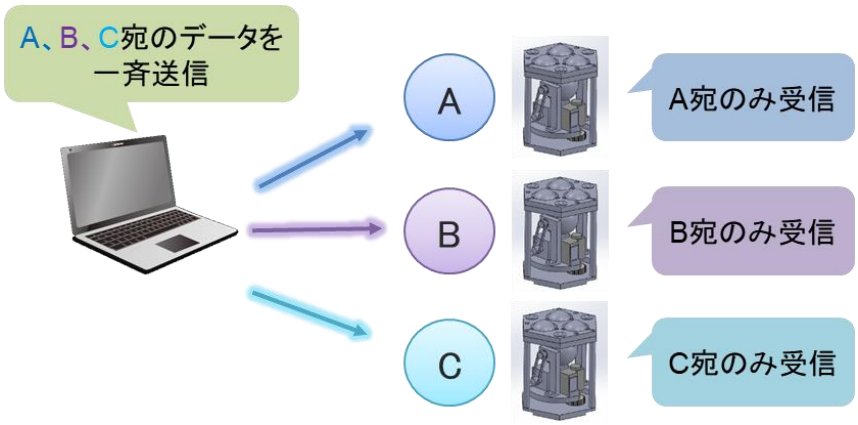


図5.4　　UDP 通信を用いたユニット制御

最初は 1 つのパソコンを使って、同じパソコン間で UDP 通信ができるかを確認した。通信のイメージを図 5.5 に示す。

図5.5　　同じ PC 間の UDP 通信のイメージ

送信側から文字列を送信し、受信側が表示しているデータが送信側と同じデータであることを確認できた。送信中に文字列の値を変更したときに受信側の値も変わったことから、1 つのパソコン間での送受信は問題ないことが分かった。次に 2 つのパソコンを用いて違うパソコン間での通信が出来るかどうか確認した。通信のイメージを図 5.6 に示す。

図5.6　異なる PC 間の UDP 通信のイメージ

同じパソコン間の制御と同じように、送信側のパソコンから文字列を送信し、受信されたデータが送信側と同じデータであることを確認した。送信中に文字列の値を変更した場合も変更後の文字列を受信できていることが確認できた。以上より 2 つのパソコン間での通信にも問題がないと判断した。次に 2 つの回路（無線通信用モジュール）を使って回路間の通信を確認した。通信のイメージを図 5.7 に示す。

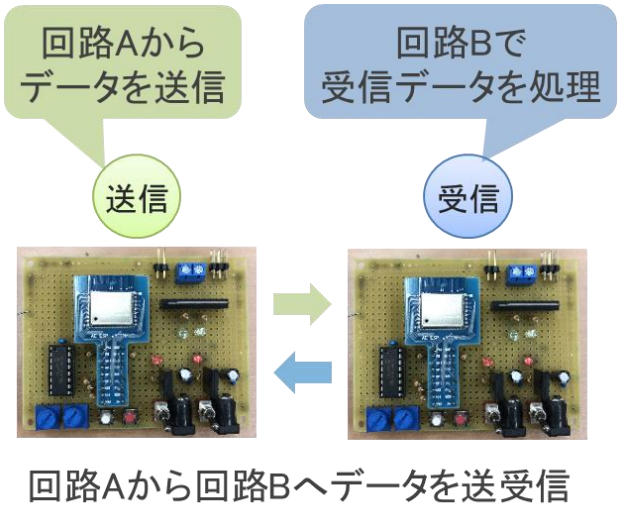


図5.7　回路間の UDP 通信のイメージ

送信側の回路でモータの出力値を変更し、受信側の回路がそのデータを受信することで、が同じになると確認できた。送信側の出力値を変更すると受信側も同じように変化したため、回路（無線通信用モジュール）間の通信ができていると分かった。パソコン間と回路間の通信に問題がなかったため、次にパソコンと回路間の通信の確認をした。通信のイメージを図 5.8 に示す。

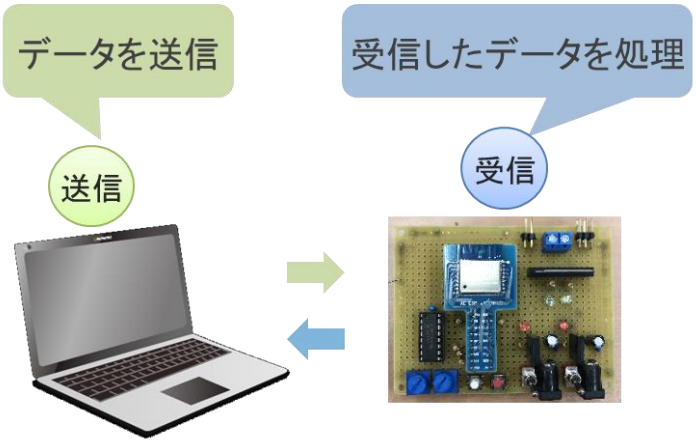


図5.8　PC と回路間の UDP 通信のイメージ

パソコンからモータの速度や角度の値を無線通信で送信し、受信した回路で処理が行われ、モータが指示通りに動作したことを確認した。パソコンから指令する値を変化させたとき、モータの動作にも指示通りの変化が見られた。このことからユニットを UDP 通信で制御しても支障がなさそうだと分かった。パソコンから 1 つの回路にデータの送受信ができることを確認できたので、受信側の回路を増やし複数ユニットに向けた制御の可否を確認することにした。通信のイメージを図 5.9 に示す。

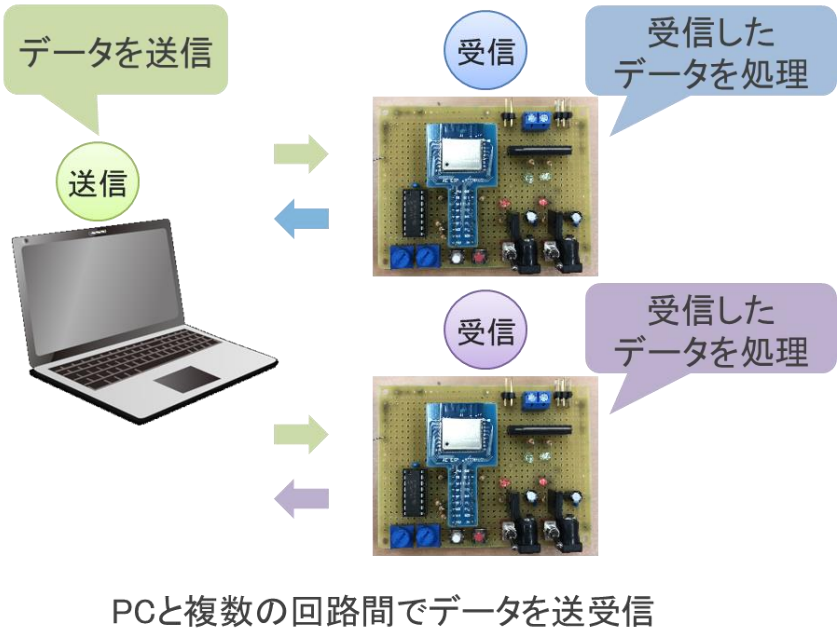


図5.9　PC と複数の回路間のデータ送受信

最初に IP アドレスとポート番号でアドレス割り振りを行い、受信する回路の識別をする。アドレス指定をしてデータを送信し、指定されたアドレスの回路がデータを処理し実行するという制御を行った。しかし、モータがパソコンの指示通りに駆動しなかった。アドレスの割り振りや指定ができていなかったことが原因ではないかと考えられる。そこでアドレスの設定が正しいかどうかを確認するため、1 つのアドレスを指定して 2 つの回路に文字列を送信して確認した。通信のイメージを図 5.10 に示す。

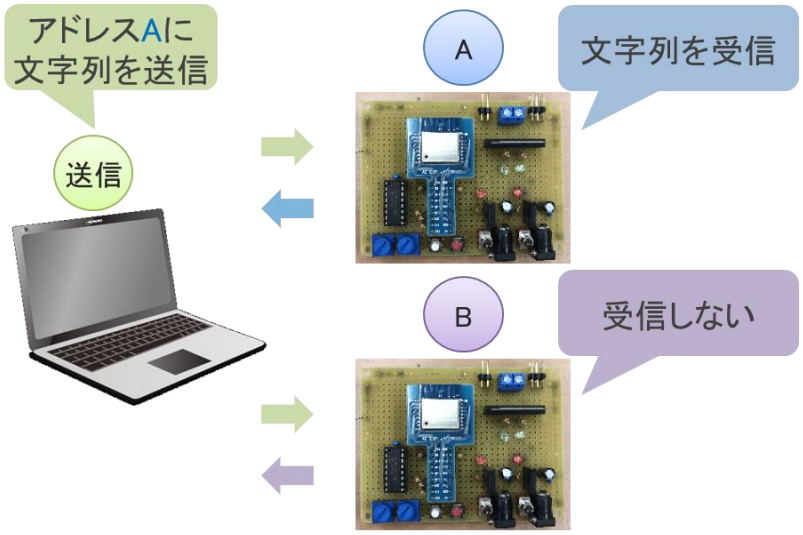


図5.10　PC と複数の回路間の文字列送受信

このとき、指定されたアドレスに文字列が送信されていることが確認できたため、アドレスの設定は問題なくできていることが分かった。しかし、文字列ではなくモータの速度や角度の値を送信すると、なぜか回路でリセットがかかってしまいモータが停止してしまった。アドレス設定の確認の際には文字列の送受信ができていたことから、データを送信した後の回路の処理に問題があるのではないかと考えたられる。さらに、リセットがかかることから、回路の容量不足が原因ではないかと推測した。そこで回路のコンデンサを容量の大きいものに変更し、再度モータの制御を行ってみたが動かなかった。原因が分からなかったため動作確認ができたところまで戻り、もう一度確認しなおすことにした。受信側の回路を減らし、パソコンと 1 つの回路でモータの制御をする図 5.8 の方法で動作確認をしなおした。Wi-Fi ルータやパソコン、動作確認をする部屋などの様々な条件を変更して動作の再確認をした結果、Wi-Fi ルータや部屋を変更すると動作しないことが新たに分かった。Wi-Fi ルータとパソコンの相性が悪かったことや、電波干渉などで通信が不安定になっていることが原因ではないかと考察する。本システムはノイズの多い工場などで使用することも考えているため、場所によっては機能しないなど、環境に左右されて通信が不安定になってしまう通信方法ではユニットに実装できないと判断した。

第6章 通信制御（有線通信）

6.1 シリアル通信とパラレル通信［7］

無線通信は、遠隔操作ができる、配線が少なく個数を増やしやすいという利点があったが、常に確実で安定した動作ができるものが必要であると考え有線通信に切り替えることにした。有線通信も無線通信の時と同じように、パソコンと回路で通信を行って制御する。パソコンを用いた一般的な通信方法について説明する。パソコンは扱うデータを全て 2 進数に変換して処理しており、2 進数のデータを送信、受信する通信方法にはシリアル通信とパラレル通信という通信方法がある。シリアル通信とは 1 本の信号線を使用して 1 ビットずつ順番にデータを伝送する方法のことである。この伝送方法の仕組みにより、伝送に時間がかかるが順番を並べなおす手間が必要ない。またクロック（タイミング）がずれないという特徴を持つ。パラレル通信にくらべて回路数が少なく低コスト、配線が簡単なため、長距離間の通信にも使われる。パラレル通信とは複数の通信回線を用いる事により、一度に複数ビットのデータを伝送する方法である。シリアル通信よりも伝達速度は速いがクロックのずれが生じやすく、複数のデータを同時に通信できるようにタイミングを考慮する必要がある。シリアル通信とパラレル通信のイメージを図 6.1 に示す。

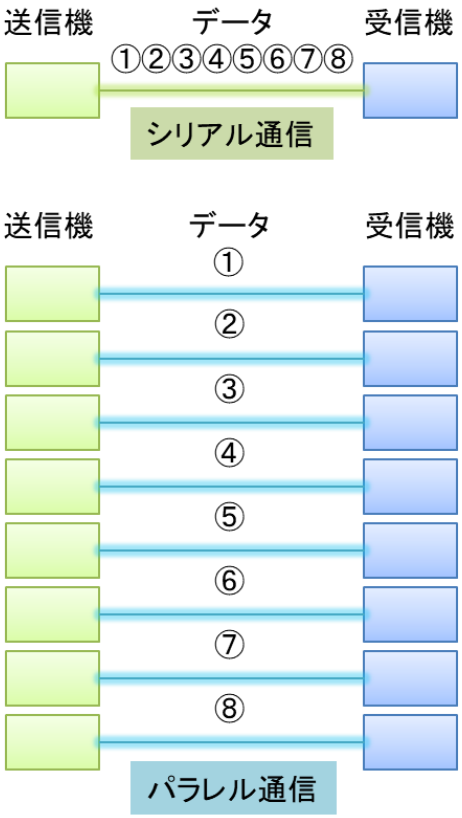


図6.1　　シリアル通信とパラレル通信のイメージ

本移動手段の制御において、配線数が多くなってしまうことや、クロックのずれが生じてしまうことは大きな欠点になると思われる。そこでクロックのずれが生じないこと、長距離間の通信が可能という点からシリアル通信を選択した。

6.2 シリアル通信を用いた制御

シリアル通信の制御目標は、部屋や環境に左右されず確実で安定した制御をすることである。制御方法は、Processing というソフトを使って、パソコンの画面に速度、角度変更操作用のノブを作成する。画面に表示されたノブを操作してモータの出力値を変更する。PC から有線（シリアル通信）で送信された数値（ノブの数値）を回路で受信する。受信データを処理してモータの角度や速度を制御するという方法である。複数のユニットを制御する場合は、分配器を使ってシリアルポートから通信を分配する。分配器の分配可能数を超えるユニット数を制御する場合は、分配器をさらに分配して補う。制御イメージを図 6.2 に示す。

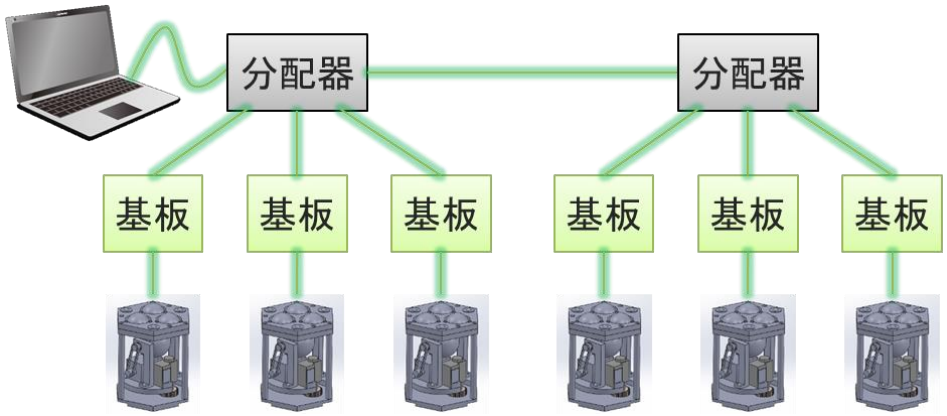


図6.2　　シリアル通信を用いたユニット制御

プログラムの作成および回路の作製をして動作実験を行った。作製した回路を図 6.3 に示す。

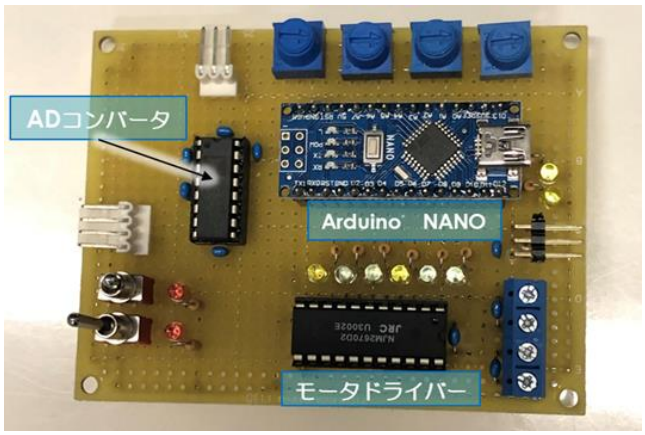
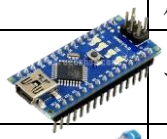
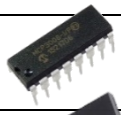
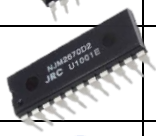


図6.3　　シリアル通信用回路

この回路を作成する際に使用した部品の仕様を表 6.1 に示す。

表6.1　　シリアル通信制御の回路に使用した部品

名称	用途、役割
Arduino　NANO 	マイコン
積層セラミックコンデンサ 	ノイズ（交流電流）を逃がす
AD コンバータ　ADM3202ANZ 	アナログ信号をデジタル信号に変換
モータドライバ NJM2670D2 	モータの制御用集積回路
ボリューム抵抗（可変抵抗器） 	つまみをまわすことで抵抗値を変えられる抵抗器

6.3　シリアル通信を用いた動作実験

2 つのユニットと 4 つのノブを準備し、複数のユニットを制御できるかを確認した。ユニット上にボールを設置して回転方向と速度の変更ができているかを確認した。動作実験の様子を図 6.4 に示す。

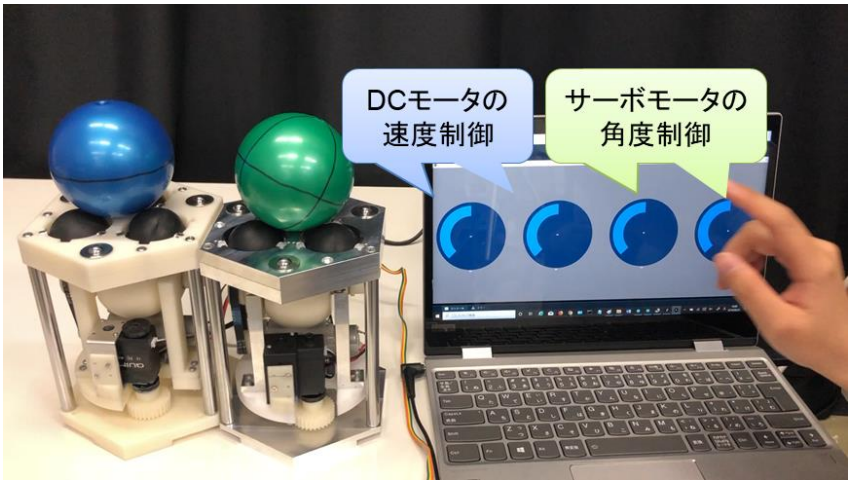


図6.4　　シリアル通信の動作実験

画面に表示されているノブの表示［8］を変更したときに、モータの動作が同期していることを確認できた。DC モータは問題なく動いたがサーボモータは振動してしまった。サーボモータの振動の原因は、プログラムの組み方に問題があると考えられる。原因と考えられるプログラムの一部をフローチャート（図 6.5）に示す。

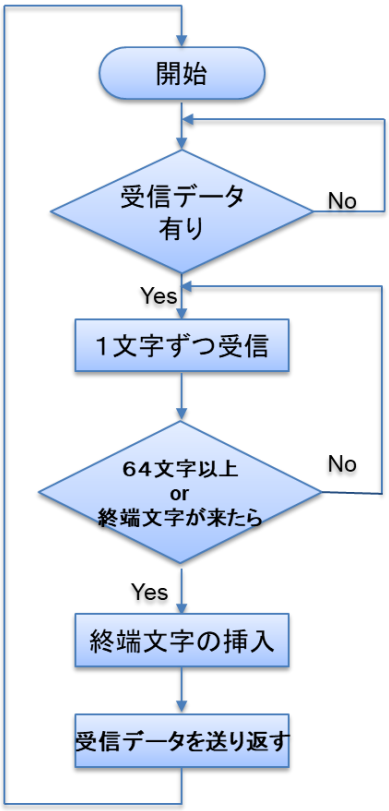


図6.5　　サーボモータ振動の原因

Processing から読み取ったノブの値を文字列に変換（A/D 変換）して Arduino で受信している。データの掃き出しなどを確認すると受信時に終端文字の処理がうまくできていないことが原因で、取得データの順番が不規則になっているのではないかと考えられるような挙動が見られた。そこで 1 文字ずつ受信せず文字列で受信するようにプログラムを変更した。またデータを常に送り返すことで情報処理量が増え負担がかかっていることも原因ではないかと想定された。そこで値が変更されたときのみ受信データを送り返すというプログラムに変更した。プログラムの見直しをした後、もう一度同じ条件で動作確認をしたところ、問題なく動作した。2 つのユニット制御ができたのでさらにユニット数を増やすことを考えてみた。しかし、この制御方法はユニットが接続された順番にしか制御ができないことが分かった。現段階での単純なプログラムでは順不同、順番飛ばしなどの制御ができないため、例えば奇数番号のみを作動させるなどの柔軟な制御ができない。また最後に接続されたユニットを作動させる場合に、それ以前のユニットも全て作動してしまうという無駄が多い制御になってしまう。他にも分配器を使って個数を増やしているため、一部の通信に問題が生じた際に、それ以降のユニットが動かないという問題点があることが発覚した。そこでこれらの課題に対応できる CAN 通信という通信方法に移行した。

6.4 CAN 通信［9］［10］

CAN(Controller Area Network)通信とは複数の CAN デバイスが互いに通信するための低コストで耐久性の高いネットワークプロトコルのことである。通信線のことをバス、ネットワークに接続される通信機器（電子制御ユニット（ECU）など）のことをノードという。複数のノードを接続してネットワークを実現する方式は、スター型、ライン型、リング型などがある。CAN はライン型の方式を採用している。通信のイメージを図 6.6 に示す。

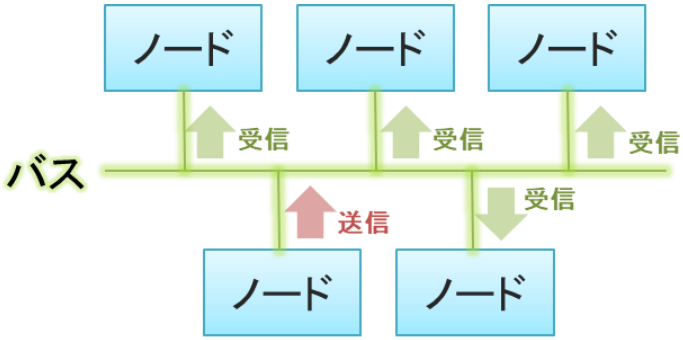


図6.6　　CAN 通信のイメージ

ライン型を採用しているので、単純にノードを接続すればネットワークが構築できるのでノードの追加や削除がしやすく、ネットワーク設計が行いやすい特徴がある。この特徴はユニットの個数や配置を変更したり、故障したユニットのみを取り換えたりすることができるようになり、有線通信であるが無線通信の時の利点も導入できる。また CAN デバイスが互いに通信することで、バスに空きがあればどの制御ユニットからでも通信を開始できる。つまりすべてのノード（電子制御ユニット（ECU））がマスターユニットとして機能できる。この特徴により、一部の通信に問題が生じた際にもそれ以降のユニットは影響を受けず、システム全体が停止することは理論的に起こりえない。以上よりシリアル通信の時の問題点が克服できたといえる。また最小限の信号線で各種のデバイスを接続できることで、有線通信の問題点である配線が多いという点も克服できるのではないかと期待できる。

6.5 CAN 通信を用いた制御

CAN 通信を用いた制御方法は 2 種類考えられる。1 つ目は多ユニットを 1 基板で制御する方法である。この方法は制御するユニット数に対して基板が少なくて済むことにより、回路設計と制御が比較的簡単になるという利点が考えられる。一方で 1 基板の故障により、その基板に接続されているユニットが動作しなくなる欠点が考えられる。そして今後センサなどの新しい機能を追加したいと考えたとき、接続するユニット数分の部品を 1 基板に収めなくてはならない。この方法では回路を変更することは厳しくなると思われる。通信方法のイメージを図 6.7 に示す。

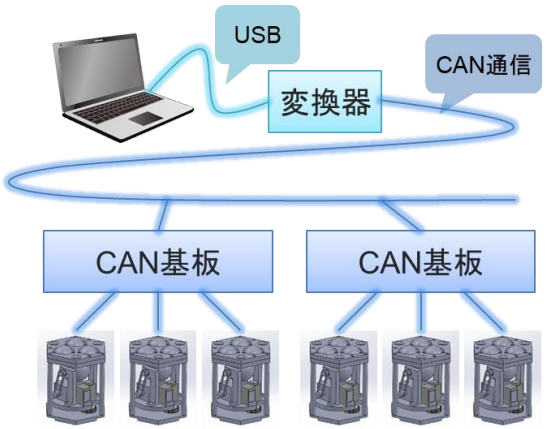


図6.7　多ユニット 1 基板の CAN 通信のイメージ

2 つ目の方法は 1 ユニットにつき 1 基板で制御する方法である。この方法ではユニット数を増やすほど回路設計が必要になる。しかし、基板の故障による複数ユニットの停止や取り変えが必要ないという利点がある。通信方法のイメージを図 6.8 に示す。

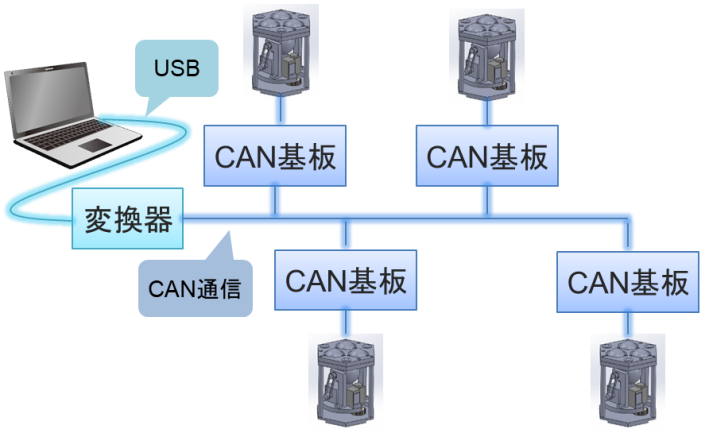


図6.8　1 ユニット 1 基板の CAN 通信のイメージ

本研究は未だ開発段階であり、今後センサなどの新しい機能を追加することを考えているため、後付けで各機能を追加しやすい後者の方法を選択した。CAN 通信を用いた制御の目標はユニットのアドレス指定を行い、複数のユニットを個別制御することである。制御方法は、Processing でパソコンの画面にアドレス番号、速度、角度変更操作用のスライダを作成する。画面に表示されたスライダを操作してモータの出力値を変更する。パソコンの USB ポートからシリアル通信で受信した出力値を変換器で CAN 通信に変換する。CAN 基板同士が互いに通信していることを確認し、変換器から CAN 通信で送信された数値（スライダの数値）を指定されたアドレス番号の回路で受信し処理をするという方法である。ユニットの番号は回路に設置したスイッチで変更する。通信方法のイメージは前述した図 6.8 である。プログラムの作成および回路の作製をして動作実験を行った。作製した回路を図 6.9 に示す。

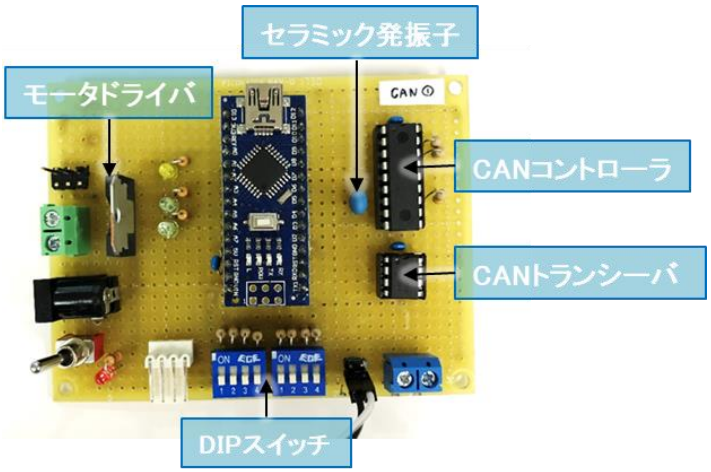


図6.9　　CAN 通信制御用回路

この回路を作成する際に使用した部品の仕様を表 6.2 に示す。

表6.2　　シリアル通信制御の回路に使用した部品

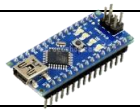
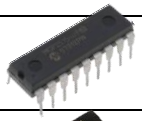
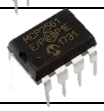
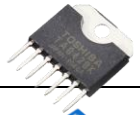
名称	用途、役割
Arduino　NANO 	マイコン
積層セラミックコンデンサ 	ノイズ（交流電流）を逃がす
CAN コントローラ　MCP2515 	CAN 通信用モジュール
CAN トランシーバ　MCP2561 	CAN 通信用モジュール
モータドライバ TA8428K 	モータの制御用集積回路
DIP スイッチ 	ユニットの番号決め
セラミック発振子 	クロック信号源

図 6.10 に示したようなスライダ［11］でアドレス番号、DC モータの速度、サーボモータの角度変更を行った。アドレスは 1 番から 10 番までとし、回路の DIP スイッチの番号とスライダの番号が一致しているときに動作する。DC モータの初期位置は 0、サーボモータの初期位置は 90 度とする。スライダの値が変化したときに各モータの速度や角度が増減するようになっている。

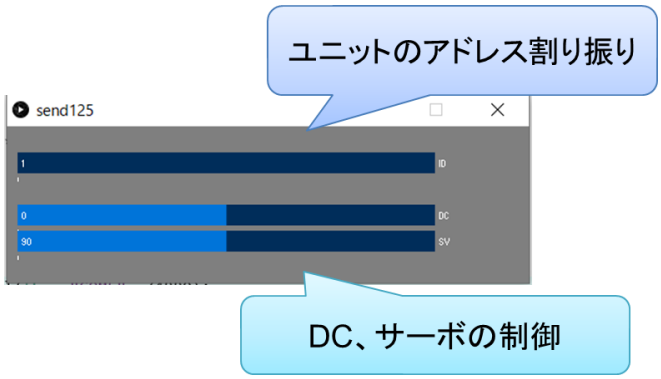


図6.10　アドレス割り用スライダとモータ制御用スライダ

加えてパソコンと変換器間、変換器と CAN 通信基板が通信できているかを確認した。動作実験の様子を図 6.11 に示す。

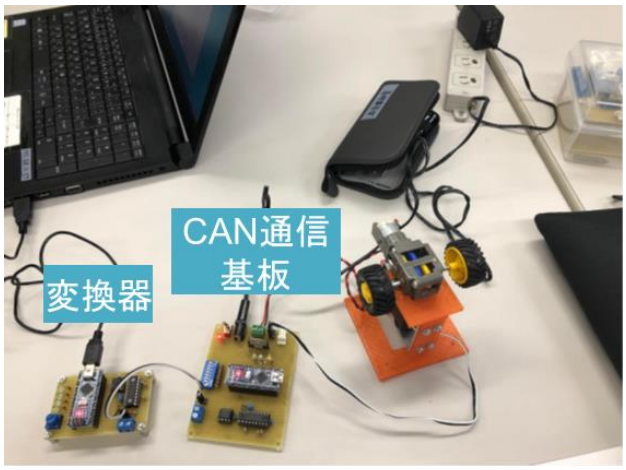


図6.11　CAN 通信の動作確認の様子

初めにユニット 1 つ分の速度制御と角度制御を行った。動作確認をしたところ、画面に表示されているスライダの表示を変更したときに、モータの動作が同期していることを確認できた。モータが正常に動作したことにより、変換器と CAN 基板も正常に動作しており、通信にも問題がないことが確認できた。次にアドレスの割り振りが正常かどうかを確認した。アドレスの番号を指定してデータを送信したとき、指定した番号と基板の番号が同じ場合は指令通りに動いた。指定するアドレス番号や基板の番号を変更し、互いの番号が違った場合は動作しないことを確認できた。このことからアドレス設定が正常に機能していることが確認できた。従って、次に複数のユニット分の動作確認に移ったところ問題点が見つかった。スライダを使ったアドレス変更は、1 ユニットずつ、順番にしかアドレス変更できな

い。例えば 1 ユニットずつしか動作できないので、1 番と 2 番を同時に動かすことができない。順番に動作するため、2 番から 5 番に番号が変更されたときに、間に配置された 3 番と 4 番のユニットも動いてしまう。また順番を変更できないことにより、1 番から 7 番まで同時制御したい場合順不同で制御ができない。以上のような問題点が分かった。そこで、アドレス割り振りはスライダではなくトグルスイッチを使うことにした。トグルスイッチを使うことで、スイッチが ON になっている番号のユニットのみ動作が可能となる。モータ制御用のスライダをユニット番号ごとに分けることで、複数のユニットを個別に制御できるようにした。トグルスイッチとスライダの様子を図 6.12 に示す。

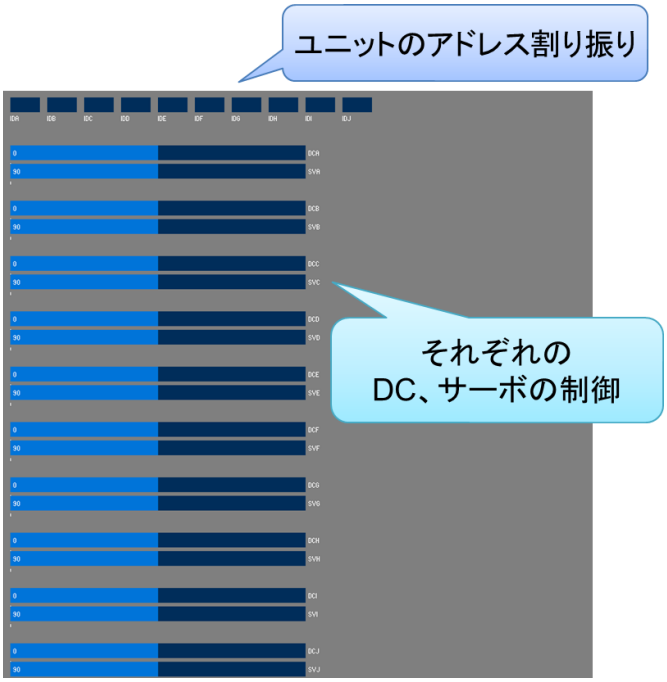


図6.12　トグルスイッチとスライダ

再度動作確認を行ったところ、アドレス指定をされているモータのみを動かせている事が確認できた。ユニット番号のトグルスイッチが OFF になっている場合、指定されたアドレスと異なるモータのスライダを操作した場合、ON になっているトグルスイッチと回路のスイッチ番号（指定されたアドレス）が異なる場合はどの条件でも動作しなかった。トグルスイッチ、回路のスイッチ番号、モータのスライダ番号がすべて一致する場合は理想通りに動いた。さらに複数の回路を接続した場合もアドレスやモータの値が交錯することなく指示通りに動作した。以上の結果から複数ユニットの個別制御ができるのではないかと判断し回路を実装し動作実験を行った。

6.6 CAN 通信を用いた動作実験

9 個のユニットに回路を実装させ動作実験を行った。動作実験の様子を図 6.13 に示す。

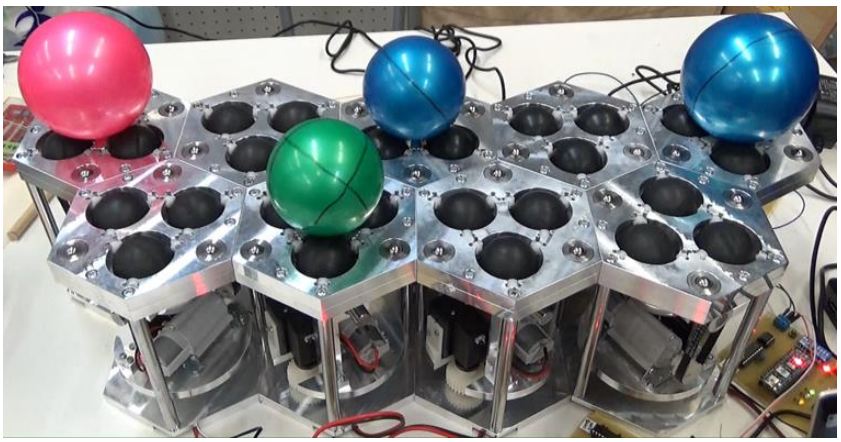


図6.13　動作実験の様子

動作実験を行った結果、複数ユニットの個別制御ができた。指定されたアドレスのユニットごとに角度や速度変更をできた。また、一部の回路の電源を OFF にすることで動作不良のユニットがある場合についても確認をした。接続されたユニットの順番に関係なく、どのユニットが動作不良の場合でもシステム全体には影響しないことが分かった。これらのことからシリアル通信を用いた場合の課題を解決できたといえる。しかし、9 個のユニットをすべて動かした状態で複数のモータを同時に制御しようとすると、動作が重くなりラグが発生してしまった。これはプログラムの処理方法が影響しているのではないかと考察する。現在は接続しているユニット全てのデータ処理を一度に行っているため、無駄な処理量が増えていることが原因であると考えられる。改善策として、プログラムを出力値に変化があったユニットのみ処理をすることや、無駄な delay（）を省くなどに変更することでラグの発生を抑えられるのではないかと考えられる。

第7章 結言

全方向移動システムを実現するために 2 種類のモータを制御するプログラムの作成と通信制御手段の選択を行った。最初に配線の手間などの削減を目的に、無線通信を用いた制御に取り組んだが電波干渉などが原因で安定な通信の維持に問題があることが明らかとなった。環境に左右されて通信が不安定になってしまう通信方法ではユニットに実装できないと判断し、有線通信に移行した。有線通信ではまずシリアル通信を使ったが、一部の通信に問題が生じた際に、それ以降のユニットが動かないという問題が発覚した。そこで一部の故障がシステム全体に影響しない CAN 通信という通信方法に移行した。CAN 通信ではアドレス設定とモータの動作確認をしたところ、複数のユニット制御ができた。今後はセンサも組み込んだ回路を実装して動作実験を行う。次に危険予測、対象物の位置把握やルート決定、自動化などを含めたより実用化を目指した制御を行っていく。

参考文献

[1] 藤川涼平，“球体伝達機構と全方向移動装置を用いた次世代移動手段の開発”，修士論文(2017)

[2] 竹中克昭，“ホイール配置による球体の全方向回転制御機構の開発”，学士論文(2017)

[3] 狩野大輝，“球体伝達機構を用いた全方向移動手段の開発”，学士論文(2018)

[4] FELLOWS　Creator’s Agent https://job.fellow-s.co.jp/useful_info/feature_detail/Art-0284

[5] 佐々木　真「分かりそう」で「分からない」でも「分かった」気になれる IT 用語辞典 UDP とは https://wa3.i-3-i.info/word110.html

[6] 佐々木　真「分かりそう」で「分からない」でも「分かった」気になれる IT 用語辞典 TCP とは https://wa3.i-3-i.info/word19.html

[7] シリアル通信の仕組み徹底解説 http://serialcomm.info/?p=4

[8] Aa develop 開発と成長　ControlP5 の使い方 http://aa-deb.hatenablog.com/entry/2016/11/04/230332

[9] mindwood 猿でも分かる CAN プロトコルの基本 https://qiita.com/mindwood/items/93d5c2dc0e340cc20769

[10]KEYENCE CAN とは？https://www.keyence.co.jp/ss/products/recorder/lab/candata/base.jsp

[11]akspect Processing controlP5 で GUI コントローラー https://qiita.com/akspect/items/6a574e12181c00125d40

付録

6.2 Web サーバの実装

```
*****************************************
//ESP-led を遠隔操作で光らせる//
*****************************************
#include <ESP8266Wifi.h>
#include<WiFiClient.h>
#include<ESP8266WebServer.h>
#include<ESP8266mDNS.h>

const chart*              ssid="0000000"
const chart*       password="0000000"

ESP8266WebServer server(80);

const int led    = 13;
const int led2 = 14;

void setup()
{
      Wifi.config(IPAddress(192, 168,11,37),
                        IPAddress(192, 168,11,100),
                        IPAddress(255, 255,0,0);
      pinMode(led, OUTPUT);
      digitalWrite(led, 0);
      Serial.begin(9600);
      WiFi.begin(ssid, password);
      Serial.println("");

      //wait for connection
      while (WiFi.status() !=WL_CONNECTED)
      {
            delay(500);
            Serial.print(".");
      }
```

```
Serial.println(".");
Serial.print("Connected to");
Serial.print(ssid);
Serial.print("IP address:");
Serial.print(WiFi.localIP());

if (MDNS.begin("esp8266"))
{
      Serial.println("MDNS responder started");
}

pinMode(led2, OUTPUT);
analogWrite(led2, 0);
server.on("/", []()
{
      String s = server.arg("IO");

      digitalWrite(13,0);
      digitalWrite(14,0);

      if (s=="led13")
      {
            digitalWrite(13,1);
      }

      if (s=="led14")
      {
            digitalWrite(14,1);
      }

      if (s=="AC")
      {
            digitalWrite(13,0);
            digitalWrite(14,0);
      }
```

```
        Serial.print(s[0]);
        Serial.print(s[1]);
        Serial.print(s[2]);
        Serial.print(s[3]);
        Serial.print(s[4]);
        Serial.println(s[5]);

        String msg = ".";
        msg   += "<html>";

        msg   += "<head>";
        msg   += "<title> Hello Servor 07 </title>";
        msg   += "</head>";

        msg   += "<body>";
        msg   += "<br>";
        msg   += "<hr>";
        msg   += "<h4> Hello Servor 07 </h4>";
        msg   += "<h2> Submit Button Sample </h2>";
        msg   += "<h4> Kochi University of Technology </h4>";
        msg   += "<hr>";

        msg   += "<form method='get' action='./'>";
        msg   += "<input type='submit' name='IO' value='LED13'>";
        msg   += "<input type='submit' name='IO' value='LED14'>";
        msg   += "<input type='submit' name='IO' value='AC'>";
        msg   += "</form>";

        msg   += "</body>";
        msg   += "</html>";

        server.send(200, "text/html", msg);
});

server.begin();
serial.println("HTTP server started");
```

```
}

void loop(void)
{
      server.handleClient();
}
```

5.4　　DC モータとサーボモータの同時制御

```
/*************************************************************************
suzu22
                                                Copyright (c) 2013-2016 Wataru KUNINO
*************************************************************************/
/*
 * D2    サーボモータ
 * D4    DC モータ in1,LED
 * D5    DC モータ in2,LED
 * D12   MCP3008_dout
 * D13   MCP3008_din
 * D14   MCP3008_clk
 * D15   MCP3008_cs
 */

#include <Servo.h>
#include <MCP3008.h>

#define CS_PIN       15
#define CLOCK_PIN 14
#define MOSI_PIN    13
#define MISO_PIN    12

MCP3008 adc(CLOCK_PIN, MOSI_PIN, MISO_PIN, CS_PIN);

Servo sm2;                  // Servo オブジェクトを宣言

void setup()
```

```
{
  pinMode(2,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);

  pinMode(12,INPUT);
  pinMode(13,OUTPUT);
  pinMode(14,OUTPUT);
  pinMode(15,OUTPUT);

  Serial.begin(9600);

  sm2.attach(2);              // sm2 は制御パルスを 2 ピンに出力
  sm2.write(90);              // sm2 の角度を指令

  // モータ停止
  digitalWrite(4,0);
  digitalWrite(5,0);

  Serial.println("start");

  digitalWrite(4,1);    digitalWrite(5,1); delay(500);
  digitalWrite(4,0);    digitalWrite(5,0); delay(500);
  digitalWrite(4,1);    digitalWrite(5,1); delay(500);
}

void loop()
{
  int      ad,as,d;
  char     m[16];

  // サーボモータ
  as= adc.readADC(0);
  d=map(as,0,1023,0,179);
  sm2.write(d);
```

```
  // DC モータ
  digitalWrite(4,0);
  ad=adc.readADC(1);
  analogWrite(5,ad);

  sprintf(m,"as=%4d ad=%3d",as,ad);
  Serial.println(m);
  delay(500);
}
```

6.4　　UDP 通信を用いた制御

```
/*****************************************************************************
    受信側プログラム  rece00      (送信側は send00)
                                                20181106 Yamamoto Toshimi
    suzu_r00
*****************************************************************************/

#include <ESP8266WiFi.h>                        // Wi-Fi 機能を利用するために必要
extern "C" {
#include "user_interface.h"                     // ESP8266 用の拡張 IF ライブラリ
}
#include <WiFiUdp.h>                            // UDP 通信を行うライブラリ

#define PIN_EN 13                               // IO 13(5 番ピン)は LED を接続

#define SSID "0024A5EAF72A_G"                   // 星野研 1 F アクセスポイントの SSID
#define PASS "ynk3km55k3rh6"                    // パスワード

#define PORT 1024                               // 受信ポート番号
WiFiUDP udp;                                    // UDP 通信用のインスタンスを定義

void setup()                                    // 起動時に一度だけ実行する関数
{
    int       wt;                               // ルータへの接続待ち変数

    pinMode(PIN_EN,OUTPUT);                     // LED 端子を出力に
```

```
    Serial.begin(74880);                        // Serial の初期化

    WiFi.mode(WIFI_STA);                        // 無線 LAN を STA モードに設定
    WiFi.begin(SSID,PASS);                      // 無線 LAN アクセスポイントへ接続

    wt=0;
    while(WiFi.status() != WL_CONNECTED)        // 接続に成功するまで待つ
    {
        delay(500);                             // 待ち時間処理
        wt=wt+1;
        digitalWrite(PIN_EN,wt%2);              // LED の点滅(wt を 2 で割った余り 0 or 1)
        Serial.print("=");                       // シリアル画面に出力
    }

    WiFi.config(IPAddress(192,168,11,31),   // 本機の IP アドレス（固定 IP）
                IPAddress(192,168,11,100), // ルーターの IP アドレス（固定 IP）
                IPAddress(255,255,0,0));    // ネットマスク

    Serial.println(WiFi.localIP());             // 本機の IP アドレスをシリアル出力
    udp.begin(PORT);                            // UDP 通信御開始
}

void loop()                                     // 繰り返し実行する関数
{
    char    c;                                  // 文字変数 c を定義
    int     len;                                // 文字列長を示す整数型変数を定義
    char    m[33];                              // 文字列変数

    Serial.print(".");                          // シリアル画面に出力
    delay(100);                                 // 待ち時間

    memset(m, 0, 33);                           // 文字列変数 lcd の初期化(33 バイト)
    len = udp.parsePacket();                    // 受信パケット長を変数 len に代入
    if (len==0) return;                         // 未受信のときは loop()の先頭に戻る

    digitalWrite(PIN_EN,HIGH);                  // LED を ON に
```

```
    udp.read(m, 48);                                // 受信データを文字列変数 lcd へ代入
    m[7]=0;                                         // 文字列の終了
    Serial.println(m);                              // シリアル画面に受信データを出力
    digitalWrite(PIN_EN,LOW);                       // LED を OFF に
}

/*********************************************************************************
    送信側プログラム　send00　　（受信は rece00）
                                                    20161209 Yamamoto Toshimi
    suzu_s00
*********************************************************************************/

#include <ESP8266WiFi.h>                            // ESP8266 用ライブラリ
extern "C" {
#include "user_interface.h"                         // ESP8266 用の拡張 IF ライブラリ
}
#include <WiFiUdp.h>                                // UDP 通信を行うライブラリ
#define PIN_EN 13                                   // IO 13(5 番ピン)をセンサ用の電源に

#define SSID "0024A5EAF72A_G"                       // 星野研１F アクセスポイント
#define PASS "ynk3km55k3rh6"                        // パスワード 192.168.11.100

#define SENDTO "192.168.11.31"                      // 送信先の IP(MPS)
#define PORT 1024                                   // 送信のポート番号

WiFiUDP udp;                                        // UDP 通信用のインスタンスを定義
char    m[16];                                      // 文字列変数

void setup()                                        // 起動時に一度だけ実行する関数
{
    int wt;                                         // ルータへの接続待ち変数

    pinMode(PIN_EN,OUTPUT);                         // センサ用の電源を出力に

    Serial.begin(74880);                            // 動作確認のためのシリアル出力開始
    Serial.println("send00");                       // 「send20」をシリアル出力表示
```

```
    WiFi.mode(WIFI_STA);                          // 無線 LAN を STA モードに設定
    WiFi.begin(SSID,PASS);                        // 無線 LAN アクセスポイントへ接続

    wt=0;
    while(WiFi.status() != WL_CONNECTED)      // 接続に成功するまで待つ
    {
        delay(500);                               // 待ち時間処理
        wt=wt+1;                                  // 待ち時間カウンタを 1 加算する
        digitalWrite(PIN_EN,wt%2);                // LED(EN 信号)の点滅　2 で割った余り
0 or 1
        Serial.print('=');                        // シリアル画面に出力
    }

    WiFi.config(IPAddress(192,168,11,32),     // 本機の IP アドレス（固定 IP）
                IPAddress(192,168,11,100),      // ルーターの IP アドレス（固定 IP）
                IPAddress(255,255,0,0));      // ネットマスク

    Serial.println(WiFi.localIP());           // 本機の IP アドレスをシリアル出力
    udp.begin(PORT);                              // UDP 通信御開始
}

void loop()                                       // 無限ループで繰り返し実行する関数
{
    int             dt;                           // アナログ入力の変数

    digitalWrite(PIN_EN,HIGH);                    // LED を ON に
    dt=system_adc_read();                         // AD 変換器から値を取得
    sprintf(m,"dt=%4d",dt);                       // 整数を文字列に変換

    udp.beginPacket(SENDTO, PORT);                // UDP 送信先を設定
    udp.println(m);                               // アナログ値を送信

    Serial.println(m);                            // シリアル出力にアナログ値を表示
    udp.endPacket();                              // UDP 送信の終了(実際に送信する)
```

```
    digitalWrite(PIN_EN,LOW);                    // LED を OFF に
    delay(1000);                                 // 待ち時間
}
```

7.3　　シリアル通信を用いた動作実験

Arduino

```
/*                  *****   suzu70.ino   20190629   *****
 *
 * D0    TX0
 * D1    RX0
 * D2
 * D3    EnaA
 * D4    INA1
 * D5    INA2
 * D6    EnaB
 * D7    INB1
 * D8    INB2
 * D9
 * D10   s_TX
 * D11   s_RX
 * D12   Servo0
 * D13   Servo1
 * A0    DC0
 * A1    DC1
 * A2    SV0
 * A3    SV1
 * A4（D18)
 * A5（D19)
 * A6（D20)
 * A7（D21)
 */

#include <SoftwareSerial.h>
SoftwareSerial S_Serial(11,10);   // RX,TX

#include   <Servo.h>
```

```
Servo       svm0,svm1;

char m[64];                                          //  文字列格納用
int i = 0;                                             //  文字数のカウンタ

void setup()
{
      pinMode(3,OUTPUT);          // EnaA
      pinMode(4,OUTPUT);          // INA1
      pinMode(5,OUTPUT);          // INA2
      pinMode(6,OUTPUT);          // EnaB
      pinMode(7,OUTPUT);          // INB1
      pinMode(8,OUTPUT);          // INB2

      pinMode(12,OUTPUT);         // Servo0
      pinMode(13,OUTPUT);         // Servo1

      svm0.attach(12);
      svm1.attach(13);

      analogWrite(3,0);     // DC0
      digitalWrite(4,1);
      digitalWrite(5,0);

      analogWrite(6,0);     // DC1
      digitalWrite(7,1);
      digitalWrite(8,0);

      Serial.begin(9600);
      S_Serial.begin(9600);
}

void loop()
{
      int dc0,dc1,sv0,sv1;
```

```
    // データ受信したとき
    if (S_Serial.available())
    {
        m[i] = S_Serial.read();                    // 1 文字ずつ受信

        // 文字数が 64 以上 or 終端文字が来たら
        if (i > 64 || m[i] == '¥0')
        {
            m[i] = '¥0';                           // 末尾に終端文字の挿入
            S_Serial.print(m);                     // 受信文字列を送信
            i = 0;                                 // カウンタの初期化

            dc0=(m[1]-0x30)*100 + (m[2]-0x30)*10 + (m[3]-0x30)*1;
            dc1=(m[4]-0x30)*100 + (m[5]-0x30)*10 + (m[6]-0x30)*1;
            sv0=(m[7]-0x30)*100 + (m[8]-0x30)*10 + (m[9]-0x30)*1;
            sv1=(m[10]-0x30)*100 + (m[11]-0x30)*10 + (m[12]-0x30)*1;

            analogWrite(3,dc0);
            analogWrite(6,dc1);

            svm0.write(sv0);
            svm1.write(sv1);
        }
        else
        {
            i=i+1;
        }
    }
}
```

Processing

```
import processing.serial.*;
Serial ser;

import controlP5.*;

ControlP5    sdc0, sdc1, ssv0, ssv1;
```

```
int                         dc0,   dc1,   sv0,   sv1;


void setup(){
   size(800, 300);

   ser = new Serial(this,"COM3", 9600);


   sdc0 = new ControlP5(this);
   sdc0.addKnob("dc0")
       .setLabel("dc0")
       .setRange(0, 255)
       .setValue(127)
       .setPosition(80, 50)
       .setRadius(50)
       .setDragDirection(Knob.VERTICAL);

   sdc1 = new ControlP5(this);
   sdc1.addKnob("dc1")
       .setLabel("dc1")
       .setRange(0, 255)
       .setValue(127)
       .setPosition(230,50)
       .setRadius(50)
       .setDragDirection(Knob.VERTICAL);

   ssv0 = new ControlP5(this);
   ssv0.addKnob("sv0")
       .setLabel("sv0")
       .setRange(0, 180)
       .setValue(90)
       .setPosition(380, 50)
       .setRadius(50)
       .setDragDirection(Knob.VERTICAL);
```

```
  ssv1 = new ControlP5(this);
  ssv1.addKnob("sv1")
      .setLabel("sv1")
      .setRange(0, 180)
      .setValue(90)
      .setPosition(530, 50)
      .setRadius(50)
      .setDragDirection(Knob.VERTICAL);


}

void draw()
{
  background(127);

  String m = 1 + nf(dc0,3) + nf(dc1,3) + nf(sv0,3) + nf(sv1,3);

  ser.write(m);
  ser.write("¥0");

  delay(100);

  if ( ser.available() > 0)
  {
    String data = ser.readString();          // 文字列を受信
    println(data);
  }
}
```

7.6 CAN 通信を用いた動作実験

Junction Arduino

```
//              *****  junc125.ino  20191217  *****
/*
 * D0
```

```
 * D1
 * D2     INT
 * D3
 * D4
 * D5
 * D6
 * D7
 * D8
 * D9
 * D10    CS
 * D11    MOSI
 * D12    MISO
 * D13    SCK
 * A0（D14）     半固定抵抗
 * A1（D15)
 * A2（D16）     LED3
 * A3（D17）     LED2
 * A4（D18）     LED1
 * A5（D19）     LED0
 * A6（D20)
 * A7（D21)
 */
#include <stdio.h>
#include <mcp_can.h>
#include <SPI.h>
#define    CAN0_INT 2                                   // Set INT to pin    2
MCP_CAN CAN0(10);         // Set CS to pin 10

char m[128];                                          //  文字列格納用
int i = 0;                                           //  文字数のカウンタ

long unsigned int    id;
unsigned char          rid;                    // rid  受信 id
unsigned char          len = 0;
char          buf[8];
```

```
void setup()
{
	Serial.begin(74880);

	pinMode(19,OUTPUT);
	pinMode(18,OUTPUT);

	if (CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_20MHZ) == CAN_OK)
		digitalWrite(19,0);
	else
		digitalWrite(19,1);

	CAN0.setMode(MCP_NORMAL);
}

//byte buf[8] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};

void loop()
{
	int idd,dc,sv;

	// データ受信したとき
	if (Serial.available())
	{
		m[i] = Serial.read();				// 1 文字ずつ受信

		// 文字数が 64 以上 or 終端文字が来たら
		if (i > 64 || m[i] == '¥0')
		{
			m[i] = '¥0';					// 末尾に終端文字の挿入
//			  Serial.print(m);				// 受信文字列を送信
			i = 0;						// カウンタの初期化

			idd=(m[ 0]-0x30)*10 + (m[ 1]-0x30)*1;

			dc=(m[ 3]-0x30)*100 + (m[ 4]-0x30)*10 + (m[ 5]-0x30)*1;
```

```
            sv=(m[ 6]-0x30)*100 + (m[ 7]-0x30)*10 + (m[ 8]-0x30)*1;

            if (m[2]=='-') dc=-dc;

            buf[0]=0;                   //  送信元 ID=0
            buf[1]=(char)dc;            // DC ﾓｰﾀの値
            buf[2]=(char)sv;            // Servo ﾓｰﾀの値
            buf[3]=0;                   // ﾘｻﾞｰﾌﾞ  未使用
            buf[4]=0;                   // ﾘｻﾞｰﾌﾞ  未使用
            buf[5]=0;                   // ﾘｻﾞｰﾌﾞ  未使用
            buf[6]=0;                   // ﾘｻﾞｰﾌﾞ  未使用
            buf[7]=0;                   // ﾘｻﾞｰﾌﾞ  未使用

            byte sndStat = CAN0.sendMsgBuf(idd, 0, 8, buf);

            if(sndStat == CAN_OK)
            {
                digitalWrite(18,0);
            }
            else
            {
                digitalWrite(18,1);
            }
        }
        else
        {
            i=i+1;
        }
    }

    // *** センサ値の受信 ***
    if (!digitalRead(CAN0_INT))                         // If CAN0_INT pin is low, read receive
buffer
    {
        CAN0.readMsgBuf(&id, &len, buf);                // Read data: len = data length, buf = data
byte(s)
```

```
        if ((id & 0x80000000) == 0x80000000)                    // 0x1FFFFFFF =>
29bit
            sprintf(m, "Extended ID: 0x%08X    DLC: %1d    Data:", (id & 0x1FFFFFFF), len);
        else
            sprintf(m, "Standard ID: 0x%04X    DLC: %1d    Data:", id, len);

//            Serial.print(m);

        if ((id & 0x40000000) == 0x40000000)            // Determine if message is a remote
request frame.
        {
            sprintf(m, " REMOTE REQUEST FRAME");
//              Serial.println(m);
        }
        else
        {
            rid=(unsigned char)(id & 0x00ff);
            if (rid==0)                                         // ｼﾞｬﾝｸｼｮﾝ基板の ID は 0
            {
                sprintf(m,"id=%d adc=%d",buf[0],buf[1]);
                Serial.println(m);
            }
        }
    }
}
```

Arduino

```
// *****   simp125.ino    10191217    *****

/*
 * D0     TX0
 * D1     RX0
 * D2     INT
 * D3     INA1
 * D4     INA2
 * D5     servo
 * D6     DipSW0
```

```
 * D7     DipSW1
 * D8     DipSW2
 * D9     DipSW3
 * D10    CS
 * D11    MOSI
 * D12    MISO
 * D13    SCK
 * A0（D14）    ﾌｫﾄｾﾝｻの値を入力
 * A1（D15）    ﾌｫﾄｾﾝｻへ電源の供給
 * A2（D16）    DipSW4
 * A3（D17）    DipSW5
 * A4（D18）    DipSW6
 * A5（D19）    DipSW7
 * A6（D20）
 * A7（D21）
 */
#include <stdio.h>
#include <mcp_can.h>
#include <SPI.h>
#define    CAN0_INT 2                                   // Set INT to pin    2
MCP_CAN CAN0(10);                                       // Set    CS to pin 10

#include    <Servo.h>
Servo       svm0;

long unsigned int    id;
unsigned char          len = 0;
char                     buf[8];
char                     m[128];                    // Array to store serial string

unsigned char          rid,sid;                   // rid  受信 id      sid  ｽｲｯﾁ id

void setup()
{
      unsigned char      s0,s1,s2,s3,s4,s5,s6,s7;
```

```
int    an0,an1;
unsigned char a0,a1;

pinMode(3,OUTPUT);          // INA1
pinMode(4,OUTPUT);          // INA2
pinMode(5,OUTPUT);          // servo

pinMode(6,INPUT);           // dipSW0
pinMode(7,INPUT);           // dipSW1
pinMode(8,INPUT);           // dipSW2
pinMode(9,INPUT);           // dipSW3

pinMode(15,OUTPUT);         // ﾌｫﾄｾﾝｻの LED 電源

pinMode(16,INPUT);          // dipSW4
pinMode(17,INPUT);          // dipSW5
pinMode(18,INPUT);          // dipSW6
pinMode(19,INPUT);          // dipSW7

svm0.attach(5);

analogWrite(3,0);           // 停止
digitalWrite(4,0);          // 正転

svm0.write(90);             // ﾎｰﾑﾎﾟｼﾞｼｮﾝ
delay(1000);

Serial.begin(74880);

if (CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_20MHZ) == CAN_OK)
    Serial.println("MCP2515 Initialized Successfully!");
else
    Serial.println("Error Initializing MCP2515...");

CAN0.setMode(MCP_NORMAL);
pinMode(CAN0_INT, INPUT);                           // CAN0_int = 2
```

```
    Serial.println("MCP2515 Library Receive Example...");

    s0=digitalRead(6);      s0=~s0 & 0x01;
    s1=digitalRead(7);      s1=~s1 & 0x01;
    s2=digitalRead(8);      s2=~s2 & 0x01;
    s3=digitalRead(9);      s3=~s3 & 0x01;
    s4=digitalRead(16);     s4=~s4 & 0x01;
    s5=digitalRead(17);     s5=~s5 & 0x01;
    s6=digitalRead(18);     s6=~s6 & 0x01;
    s7=digitalRead(19);     s7=~s7 & 0x01;

    sid=s7*128+s6*64+s5*32+s4*16+s3*8+s2*4+s1*2+s0*1;

    sprintf(m,"%1d %1d %1d %1d %1d %1d %1d %1d",s7,s6,s5,s4,s3,s2,s1,s0);
    Serial.println(m);
}

void loop()
{
    int         dc,sv;
    int         an0;
    unsigned int an;

    if (!digitalRead(CAN0_INT))
    {
        CAN0.readMsgBuf(&id, &len, buf);

        // Determine if ID is standard (11 bits) or extended (29 bits)
        if ((id & 0x80000000) == 0x80000000)                    // 0x1FFFFFFF =>
29bit
            sprintf(m, "Extended ID: 0x%08X  DLC: %1d  Data:", (id & 0x1FFFFFFF), len);
        else
            sprintf(m, "Standard ID: 0x%04X  DLC: %1d  Data:", id, len);
        Serial.print(m);
```

```
if ((id & 0x40000000) == 0x40000000)
{
    sprintf(m, " REMOTE REQUEST FRAME");
    Serial.println(m);
}
else
{
    rid=(unsigned char)(id & 0x00ff);
    sprintf(m, "    sid=%4d rid=%4d => ",sid,rid);
    Serial.print(m);

    // 整数データ８個をシリアルモニタに表示
    for(byte i = 0; i<len; i++)
    {
        sprintf(m, "%4d", buf[i]);
        Serial.print(m);
    }
    Serial.println(" ");

    if (sid == rid)
    {
        // サーボモータの角度は半分で送られてくるので 2 倍する。
        dc=buf[1]*2;                    // DC モータの PWM 値
        sv=buf[2]*2;                // サーボモータの角度

        if (dc>=0)                  // 正転
        {
            analogWrite(3,dc);
            digitalWrite(4,0);
        }
        else                    // 反転
        {
            dc=-dc;
            dc=255-dc;
            analogWrite(3,dc);
            digitalWrite(4,1);
```

```
                    }

                    svm0.write(sv);

                    // ***  センサ値を junction 基板へ送る      ***

                    digitalWrite(15,1);
                    delay(10);
                    an0=analogRead(0);                    // センサ値の入力      10bit
                    digitalWrite(15,0);

                    buf[0]=(char)sid;                     // 送り元 ID
                    buf[1]=(char)(an0 >> 4);              // 6bit センサ値  0-63
                    buf[2]=0;
                    buf[3]=0;
                    buf[4]=0;
                    buf[5]=0;
                    buf[6]=0;
                    buf[7]=0;
                    byte sndStat = CAN0.sendMsgBuf(0,0,8,buf);   // ｼﾞｬﾝｸｼｮﾝ基板(id=0)へ送信
                }
            }
        }
        delay(100);
}
```

Processing

```
//        *******    send125.pde 20191217 *******

import processing.serial.*;
Serial ser;

import controlP5.*;
ControlP5                 cp5,sid,sd0,sd1;
int                        id,dc, sv, sv2,dc2,
                             ida,dca,sva,sv2a,dc2a,
```

```
                    idb,dcb,svb,sv2b,dc2b,
                    idc,dcc,svc,sv2c,dc2c,
                    idd,dcd,svd,sv2d,dc2d,
                    ide,dce,sve,sv2e,dc2e,
                    idf,dcf,svf,sv2f,dc2f,
                    idg,dcg,svg,sv2g,dc2g,
                    idh,dch,svh,sv2h,dc2h,
                    idi,dci,svi,sv2i,dc2i,
                    idj,dcj,svj,sv2j,dc2j;

boolean
toggleaValue,togglebValue,togglecValue,toggledValue,toggleeValue,togglefValue,togglegValue,toggl
ehValue,toggleiValue,togglejValue;
Toggle toggle;

void setup()
{
  size(800, 810);
  ser = new Serial(this,"COM5", 74880);
      //a
      cp5 = new ControlP5(this);

      cp5.addToggle("toggleaValue")
      .setLabel("ida")
      .setPosition(10, 10)
      .setValue(false)
      .setSize(40, 20);

      //b
      cp5 = new ControlP5(this);

      cp5.addToggle("togglebValue")
      .setLabel("idb")
      .setPosition(60, 10)
      .setValue(false)
      .setSize(40, 20);
```

```
//c
cp5 = new ControlP5(this);

cp5.addToggle("togglecValue")
.setLabel("idc")
.setPosition(110, 10)
.setValue(false)
.setSize(40, 20);

//d
cp5 = new ControlP5(this);

cp5.addToggle("toggledValue")
.setLabel("idd")
.setPosition(160, 10)
.setValue(false)
.setSize(40, 20);

//e
cp5 = new ControlP5(this);

cp5.addToggle("toggleeValue")
.setLabel("ide")
.setPosition(210, 10)
.setValue(false)
.setSize(40, 20);

//f
cp5 = new ControlP5(this);

cp5.addToggle("togglefValue")
.setLabel("idf")
.setPosition(260, 10)
.setValue(false)
.setSize(40, 20);
```

```
//g
cp5 = new ControlP5(this);

cp5.addToggle("togglegValue")
.setLabel("idg")
.setPosition(310, 10)
.setValue(false)
.setSize(40, 20);

//h
cp5 = new ControlP5(this);

cp5.addToggle("togglehValue")
.setLabel("idh")
.setPosition(360, 10)
.setValue(false)
.setSize(40, 20);

//i
cp5 = new ControlP5(this);

cp5.addToggle("toggleiValue")
.setLabel("idi")
.setPosition(410, 10)
.setValue(false)
.setSize(40, 20);

//j
cp5 = new ControlP5(this);

cp5.addToggle("togglejValue")
.setLabel("idj")
.setPosition(460, 10)
.setValue(false)
.setSize(40, 20);
```

```
  /*sid = new ControlP5(this);                  // ユニット
  sid.addSlider("id")
          .setRange(1, 8)                      //1~8 の間
          .setValue(0)                          //初期値
          .setPosition(10, 25)                 //位置
          .setSize(400, 20)                     //大きさ
          .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
*/

  //a
  sd0 = new ControlP5(this);                  // DC モータ
  sd0.addSlider("dca")
          .setRange(-127,127)                  //-127~127 の間
          .setValue(0)                         //初期値
          .setPosition(10, 75)                 //位置
          .setSize(400, 20)                     //大きさ
          .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

  sd1 = new ControlP5(this);                  //  サーボモータ
  sd1.addSlider("sva")
          .setRange(0, 180)                    //0~180 の間
          .setValue(90)                         //初期値
          .setPosition(10,100)                 //位置
          .setSize(400, 20)                     //大きさ
          .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


 //b
 sd0 = new ControlP5(this);                  // DC モータ
 sd0.addSlider("dcb")
          .setRange(-127,127)                  //-127~127 の間
          .setValue(0)                         //初期値
          .setPosition(10, 150)                //位置
          .setSize(400, 20)                     //大きさ
          .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
```

```
sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svb")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                        //初期値
        .setPosition(10,175)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


 //c
 sd0 = new ControlP5(this);                  // DC モータ
 sd0.addSlider("dcc")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 225)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

 sd1 = new ControlP5(this);                  //  サーボモータ
 sd1.addSlider("svc")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                        //初期値
        .setPosition(10,250)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


 //d
 sd0 = new ControlP5(this);                  // DC モータ
 sd0.addSlider("dcd")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 300)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
```

```
sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svd")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                        //初期値
        .setPosition(10,325)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

//e
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dce")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 375)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("sve")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                        //初期値
        .setPosition(10,400)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


//f
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dcf")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 450)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
```

```
sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svf")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                          //初期値
        .setPosition(10,475)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


//g
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dcg")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 525)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svg")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                          //初期値
        .setPosition(10,550)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


//h
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dch")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 600)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
```

```
sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svh")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                          //初期値
        .setPosition(10,625)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


//i
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dci")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 675)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値

sd1 = new ControlP5(this);                  //  サーボモータ
sd1.addSlider("svi")
        .setRange(0, 180)                    //0~180 の間
        .setValue(90)                          //初期値
        .setPosition(10,700)                 //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値


//j
sd0 = new ControlP5(this);                  // DC モータ
sd0.addSlider("dcj")
        .setRange(-127,127)                  //-127~127 の間
        .setValue(0)                         //初期値
        .setPosition(10, 750)                //位置
        .setSize(400, 20)                    //大きさ
        .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
```

```
    sd1 = new ControlP5(this);                  //  サーボモータ
    sd1.addSlider("svj")
            .setRange(0, 180)                    //0~180 の間
            .setValue(90)                        //初期値
            .setPosition(10,775)                 //位置
            .setSize(400, 20)                    //大きさ
            .setNumberOfTickMarks(1024); //Range を(引数の数-1)で割った値が 1 メモリの値
}

void draw()
{
   String m;
    background(127);
    stroke(255);

    sv2a=sva/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2a=dca/2;

    sv2b=svb/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2b=dcb/2;

    sv2c=svc/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2c=dcc/2;

    sv2d=svd/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2d=dcd/2;

    sv2e=sve/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2e=dce/2;

    sv2f=svf/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2f=dcf/2;

    sv2g=svg/2;       // signed char  では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
    dc2g=dcg/2;
```

```
  sv2h=svh/2;       // signed char では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
  dc2h=dch/2;

  sv2i=svi/2;       // signed char では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
  dc2i=dci/2;

  sv2j=svj/2;       // signed char では整数部は 0-127 で 180 ではｵｰﾊﾞｰ。ゆえに半分とする
  dc2j=dcj/2;


  //a
  if(toggleaValue==true)
  {fill(255);
  if (dca<0)
      {m = nf(1,2)            + nf(dca,3) + nf(sv2a,3);}
  else
      {m = nf(1,2) + "+" + nf(dca,3) + nf(sv2a,3);}

       ser.write(m);
       ser.write("¥0");
       delay(100);
  }

 if(toggleaValue==false)
 {noFill();}


//b
if(togglebValue==true)
  {fill(255);
  if (dcb<0)
      {m = nf(2,2)            + nf(dcb,3) + nf(sv2b,3);}
  else
      {m = nf(2,2) + "+" + nf(dcb,3) + nf(sv2b,3);}

       ser.write(m);
```

```
        ser.write("¥0");
        delay(100);
  }

if(togglebValue==false)
{noFill();}


//c
if(togglecValue==true)
  {fill(255);
  if (dcc<0)
       {m = nf(3,2)             + nf(dcc,3) + nf(sv2c,3);}
  else
       {m = nf(3,2) + "+" + nf(dcc,3) + nf(sv2c,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }

if(togglecValue==false)
{noFill();}

//d
if(toggledValue==true)
  {fill(255);
  if (dcd<0)
       {m = nf(4,2)             + nf(dcd,3) + nf(sv2d,3);}
  else
       {m = nf(4,2) + "+" + nf(dcd,3) + nf(sv2d,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }
```

```
if(toggledValue==false)
{noFill();}

//e
if(toggleeValue==true)
  {fill(255);
  if (dce<0)
       {m = nf(5,2)              + nf(dce,3) + nf(sv2e,3);}
  else
       {m = nf(5,2) + "+" + nf(dce,3) + nf(sv2e,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }

if(toggleeValue==false)
{noFill();}

//f
if(togglefValue==true)
  {fill(255);
  if (dcf<0)
       {m = nf(6,2)              + nf(dcf,3) + nf(sv2f,3);}
  else
       {m = nf(6,2) + "+" + nf(dcf,3) + nf(sv2f,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }

if(togglefValue==false)
{noFill();}
```

```
//g
if(togglegValue==true)
  {fill(255);
  if (dcg<0)
       {m = nf(7,2)              + nf(dcg,3) + nf(sv2g,3);}
  else
       {m = nf(7,2) + "+" + nf(dcg,3) + nf(sv2g,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }

if(togglegValue==false)
{noFill();}

//h
if(togglehValue==true)
  {fill(255);
  if (dch<0)
       {m = nf(8,2)              + nf(dch,3) + nf(sv2h,3);}
  else
       {m = nf(8,2) + "+" + nf(dch,3) + nf(sv2h,3);}

        ser.write(m);
        ser.write("¥0");
        delay(100);
  }

if(togglehValue==false)
{noFill();}

//i
if(toggleiValue==true)
  {fill(255);
  if (dci<0)
```

```
      {m = nf(9,2)              + nf(dci,3) + nf(sv2i,3);}
  else
      {m = nf(9,2) + "+" + nf(dci,3) + nf(sv2i,3);}

       ser.write(m);
       ser.write("¥0");
       delay(100);
  }

if(toggleiValue==false)
{noFill();}

//j
if(togglejValue==true)
  {fill(255);
  if (dcj<0)
      {m = nf(10,2)             + nf(dcj,3) + nf(sv2j,3);}
  else
      {m = nf(10
   ,2) + "+" + nf(dcj,3) + nf(sv2j,3);}

       ser.write(m);
       ser.write("¥0");
       delay(100);
  }

if(togglejValue==false)
{noFill();}




  if ( ser.available() > 0)
  {
       String data = ser.readString(); //  文字列を受信
       println(data);
```

```
    }
    }

  /* void mousePressed(){
   //マウス座標が正方形の範囲内の時
   //if(mouseX>=200 && mouseX<=250 && mouseY>=100 && mouseY<=150){
     if(toggle1Value==false){ //クリックする前がオフの時
       fill(0,255,0);
       flag=true;        //クリックしたらオンの状態に変更する
     }else{                 //クリックする前がオンの時
       fill(255,0,0);
       flag=false;       //クリックしたらオフの状態に変更する
     }
   }
}
*/


     /*if(toggle1Value){
     fill(255);
   } else {
     noFill();
   }
   rect(10, 250, 100, 100);

   if(toggle2Value){
     fill(255);
   } else {
     noFill();
   }
   rect(130, 250, 100, 100);
   */
```

謝辞

本研究は高知工科大学大学院 工学研究科 基盤工学専攻 知能機械システム工学コース材料革新サスティナブルテクノロジー研究室において、川原村敏幸教授のもとで行われました。

研究当初は制御分野に苦手意識が強く、研究が進めていけるか不安になる事もありました。研究が思うように進まなかった時に優しく適切なご指導をしてくださいました。本研究を通して難しい分野にめげずに取り組むことや新しいことに挑戦することの重要性について改めて気づくことができました。心より感謝申し上げます。

教育講師の山本利水先生は、制御初心者で途方に暮れていた私に 1 から丁寧に教えて下さいました。学部 3 年生の時から 4 年間も本当にお世話になりました。苦手意識の強かった制御にも新しく挑戦することができ、回路、はんだ付け、通信、制御プログラムなど自身の分野だけでは得られなかった知識を数多く得ることができました。お陰様で苦手意識がなくなり、他分野の知識の習得が自信へとつながりました。この 4 年間で教えて頂いたことや得られた貴重な経験は、今後も私の励みになると思います。深く感謝申し上げます。

研究室の先輩である佐藤翔太さんは論文やスライドのご指導をして頂きました。明るく話しかけやすい人柄で相談がしやすかったため、研究以外でもたくさんお世話になりました。感謝申し上げます。

共に同じ研究を行ってきた狩野大輝君、石井和磨君、和仁原季也君、松阪康永君、亀岡正樹君、高井友輝君は本当にたくさんのサポートをして頂きました。壁にぶつかったときに一緒に考え、支えてくれるメンバーがいたことはとても心強かったです。相談に乗ってくださったり、意見やアドバイスを頂いたりしたため、前向きな気持ちで研究を進めることができました。皆さんの支えが無ければ目標のところまで進めることはできませんでした。本当にありがとうございました。

回路作りを手伝ってくださった石井和磨君、尾﨑珠子さん、安岡龍哉君、亀岡正樹君は忙しい中、貴重な時間を割いてくださりありがとうございました。全員が初心者とは信じられないほどに、はんだ付けが繊細で綺麗な仕上がりだったのでとても驚きました。皆さんの力がなければ研究をここまで進めることができませんでした。感謝しています。

当研究室に所属の Dang Thai Giang さん、劉麗さん、佐藤翔太さん、上田真理子さん、岡田雄哉君、狩野大輝君、秦暦君、宮地啓太君、田頭侑貴君、石井和磨君、尾﨑珠子さん、安岡龍哉君、和仁原季也君、朝子幹太君、石川祐奈さん、福江雅さん、松坂康永君、宮田翔生君、市川怜司君、亀岡正樹君、川西善郎君、草下圭太君、高井友輝君、須佐美大夢君は研究会や勉強会で発表する際、たくさんの意見やアドバイスを頂きました。また皆さんのお陰で楽しく充実した研究室生活を送ることができました。ありがとうございました。

最後にいつでも一番の味方で支えてくれていた両親と兄弟に心より感謝申し上げます。お陰で有意義で充実した学生生活を送ることができました。社会人になってからも、この 6 年間の貴重な経験を糧に日々精進し成長していきたいと思います。