2023（令和 5）年度　　修士学位論文

# 全方向搬送装置ユークリーターの
# 自律制御システムの開発

# Development of autonomous control system
# for omnidirectional transport device Euclitor.

2024 年 3 月 13 日

高知工科大学大学院　工学研究科基盤工学専攻
知能機械工学コース

1265011 澤田　陸斗

指導教員　　川原村　敏幸

# 目次


第 1 章　緒言 ..... 1
1.1.　研究背景 ..... 1
1.2.　ユークリーターとは ..... 2
1.3.　類似システム ..... 5
1.4.　研究目標 ..... 7
第 2 章　先行研究 ..... 8
2.1.　球体型 ..... 8
2.1.1.　球体型 1 号機 ..... 9
2.1.2.　球体型 2 号機 ..... 10
2.1.3.　球体型 3 号機 ..... 12
2.2.　ベルト型 ..... 14
2.2.1.　ベルト型 1 号機 ..... 14
2.2.2.　ベルト型 2 号機 ..... 16
2.2.3.　ベルト型 3 号機 ..... 17
2.2.4.　ベルト型 4 号機 ..... 19
2.3.　通信制御 ..... 21
2.3.1.　無線通信 ..... 21
2.3.2.　有線通信制御（シリアル通信） ..... 23
2.3.3.　有線通信制御（CAN 通信） ..... 24
2.3.4.　ユニットのアドレス設定と制御方法 ..... 26
第 3 章　太陽光発電システムの開発 ..... 29
3.1.　空きスペースの利用 ..... 29
3.2.　鉛蓄電池への充電を目標とした回路設計 ..... 30
3.3.　安定化電源を用いた充放電の確認 ..... 31
3.4.　太陽光による充電 ..... 33
3.5.　太陽光パネルの電力出力特性調査 ..... 34
3.6.　太陽光発電プログラムの開発 ..... 38
3.7.　一時的な電力不足の穴埋め ..... 39
第 4 章　通信制御システムの改良 ..... 41
4.1.　従来の通信システム ..... 41
4.2.　改良案 ..... 41
4.3.　中継器作製 ..... 42
4.4.　動作実験 ..... 43
4.5.　データチェック ..... 44

第 5 章　自動制御システム.....46
5.1.　センサの搭載.....46
5.1.1.　センサの選別.....46
5.1.2.　センサの組み込み構造.....47
5.2.　制御プログラム.....49
5.3.　搬送実験.....52
5.4.　まとめ.....54
第 6 章　結言.....55
参考文献.....56
付録.....58
謝辞.....79

# 第1章　緒言

## 1.1. 研究背景

我々は社会生活の中で、あらゆる搬送手段を利用しており、現在ではニッチな需要に対しても応えられるまでに搬送用の装置は日々進化し続けている。ところで我々は搬送手段をどのように選択しているだろうか。移動距離・搬送速度・搬送重量・安定性などの異なる様々な選択肢から、ニーズに最も合っている搬送手段を選択する。例えば、長距離移動であれば車・電車・飛行機が挙げられるが、人数が少ない場合は常用車、大人数になるとバス・電車などを選択することになり、移動時間を縮める場合には新幹線・飛行機といった具合に手段を選択している。搬送手段の選択基準として、搬送距離と輸送コストに注目して、各移動・運搬手段の活躍領域を分類したものが図 1.1 である。この図によると、いまだに徒歩圏内での低コストな移動・運搬手段はまだまだ発展途上であると言える。

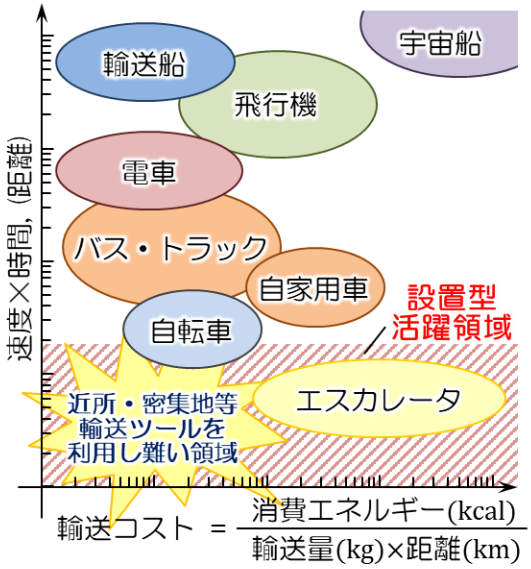


**図 1.1 各移動・運搬手段の活躍領域**

また、装置の形態について着目すれば現行移動手段は大きく二分することができ、当研究室ではそれぞれ移動型と設置型と呼んでいる。移動型は搬送するための装置（いわゆる台車）と搬送したい対象が一体となって移動をする搬送装置のことであり、車や飛行機、AGV（無人搬送車）や AMR（自律走行搬送ロボット）などが挙げられる。移動型の長所として特に固定された軌道をあまり必要とせず、あらゆる方向に搬送することができる搬送自由度の高さがある。その反面、装置に組み込まれていない外的要因が搬送システムに強く影響を及ぼすため、正確な外部刺激のインプットとそれに対応する適切なアウトプットが求められ、安

全性を確保しようとすると莫大なコスト（開発コスト・情報コストを含める）がかかる。設置型は搬送したい対象物を搬送したい（もしくはする可能性のある）領域全面に搬送する為の軌道や装置を設置し、搬送対象物のみ、もしくはそれを安全に移動させるための台車などを運搬する形態であり、エスカレーターやベルトコンベア、電車などもこちらに含まれる。設置型では搬送物が搬送される領域の搬送環境そのものから構築するため、システムに影響を及ぼす外的要因が少なく、危険予測が容易となり、高い安全性を持つ。しかし、多くの場合は搬送環境に合わせたオーダーメイドであり、初期コストが高い上に、搬送環境の変化に合わせて柔軟な対応をすることが難しい。

本研究は、従来の搬送システムである移動型・設置型、両システムの特徴を考慮し、徒歩圏内という比較的狭い範囲に利用できる、機械自身による自動制御により全方向に搬送可能かつ環境に合わせた柔軟な対応を行える移動・運搬支援ツールの構築を対象とし、新移動手段「ユークリーター」を提案した。

## 1.2. ユークリーターとは

ユークリーター（Euclitor）とは、必要領域に搬送用ユニットを敷き詰め、その敷き詰められた装置の上に搬送対象物を乗せ、その搬送対象物のみをあらゆる方向に搬送することの出来る自動搬送装置である。

ここに簡潔にユークリーターの特徴をまとめる。

1. 全方向に搬送可能
   全方向に搬送を可能とする装置構造については後述する。
2. 小型装置の集合システム
   量産した小型の搬送用ユニットを必要領域に敷き詰める使用であり、利用場所に合わせてオーダーメイドをする必要がなく、あらゆる広さ・形に柔軟に対応させることができ、後からでも容易に使用範囲の変更をすることが出来る。また、モノを運搬することと、モノを置いておくための保管場所の両方の役目を果たせるため、倉庫などの保管施設において、従来とは全く異なる新しい利用方法を提供することも期待される。
3. 必要最小ユニットのみ駆動
   使用時は機器全てを稼働させる必要はなく、搬送に必要な最小限のユニットのみ稼働させることが出来る。加えて、搬送機器自体を移動させる必要がなく、搬送対象物のみを搬送すればよい。すなわち、搬送物を搬送するために使用するエネルギーはもちろん搬送対象物を移動するためのエネルギーに限られるため、従来のシステムと比べて遙かに小さな値になる。

4. 設置型固有のシステムの制御方法を採用

現行の移動手段の大半は「他点起発事象予測方式」であり、移動対象物が周囲の状況を把握し、「いつ」「どこから」「どのようなものが」「どのような状態でくるか」などを予測しながら、危険を回避するように操縦をする方法である。この予測方式の概念図を図 1.2 に示す。

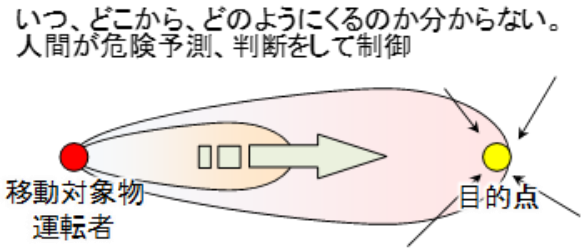


**図 1.2 他点起発事象予測方式**

しかし、この手法では制御対象外の周囲環境から危険を予測する必要があり、車の自動運転などがそうであるように、周囲情報の取得自体が難しく、常にあらゆる可能性を予測する必要があるため、高い安全性を確保することが困難である。一方、ユークリーターのシステムの制御には、周囲環境をシステム自身が構築する設置型の利点を生かし、「自点起発事象予測方式」を採用することが出来る。自点起発事象予測方式とは、システム自身が周囲の状況を把握した状態で対象物の搬送を行う手法である。この手法であれば、次の瞬間にシステム上でどのようなことが起こるかの予測することが容易で、高い安全性を確保できる。この予測方式の概念図を図 1.3 に示す。

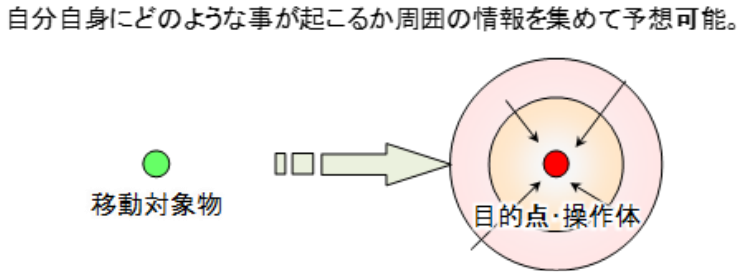


**図 1.3 自点起発事象予測方式**

ユークリーターは 1997 年に公開された「映画ドラえもん　のび太と銀河超特急」に登場する秘密道具である「ベアリングロード」から着想を得たものである。作中に登場するベアリングロードは、細かな球体が床面に敷き詰められており、念じることで球体が回転し、搭乗者を自由に移動させるという装置である。当研究室ではこのベアリングロードを現行技

術で実現させることを試み、開発を行ってきた。2020 年からこのシステムをユークリーターと名付けた。

ユークリーター（ユークリータ®）は点・直線・平面などの空間の基礎的な概念を体形化した「ユークリッド幾何学」と、エスカレーターやエレベーターといった移動手段の語尾に使われている「行為者」を表す接尾の「or」の二つを掛け合わせて作り出された造語であり、英語では Euclitor と記述する。

ユークリーターの実装は段階的に行われることが想定されている。その構想を図 1.4 に示す。まずは第 1 世代として主に物流に関わる業界を対象とした工場や倉庫での荷物の搬送から始める。第 2 世代では病院や介護施設など限定された屋内での人の歩行支援ツールとして展開する。第 3 世代ではショッピングモールや空港などの広い範囲での移動ツールとして展開し、第 4 世代では屋外での運用を想定している。また、さらにその先の世代として月や火星など重力が異なったり、水中など粘性が異なり、人が「普通」に歩いたりモノを輸送するのが困難なあらゆる状況へ適応できるシステムとして確立させることを目標としている。


**図 1.4 ユークリーターの実装構想**

## 1.3. 類似システム

ユークリーターと同じコンセプトである、必要領域に搬送用ユニットを敷き詰め、その敷き詰められた装置の上に搬送対象物を乗せ、その搬送対象物のみをあらゆる方向に搬送することの出来る機械制御システムの開発は現在世界的に注目されている。本節では特にユークリーターとシステムが類似している、複数の小型装置の敷き詰めによって搬送システムを構築する 3 つの装置を紹介する。紹介する 3 つの装置の他にも xplanar[1]というドイツの beckhoff 社が開発した搬送装置があるが、こちらは磁気浮上によって搬送を行う装置であり、ユークリーターとは少し異なるため本節において言及はしない。

1. Celluveyor（cellumation）[2]
   Celluveyor はユニット上面に取り付けられた独立して駆動させられる 3 つのオムニホイールによって搬送対象物の搬送し、オムニホイールの回転の組み合わせによって搬送方向の設定を行っている。主に工場などでの荷物の仕分け作業に利用されている。Celluveyor はその構造上 1 ユニットに付き少なくとも 3 つの動力源が必要となる。

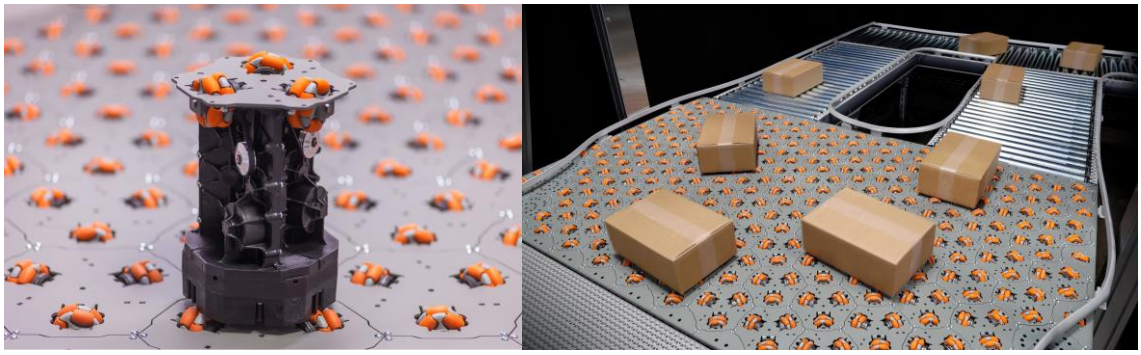

**図 1.5　Celluveyor**

2. マジックステージデバイス（伊藤電気）[3]
   マジックステージデバイスはユニット正面に取り付けられた 3 つのローラーによって搬送を行っており、上面の円形部を回転させることでローラーの向きを変更し、全方向への搬送に対応している。こちらも Celluveyor と同様、主に工場などでの荷物の仕分け作業に利用されている。マジックステージデバイスはギアを上手く組み合わせることで 2 つの動力源により搬送と搬送方向の決定を行っている。また本技術の特許出願日は本研究で取り扱うユークリーターとほぼ同時期である。

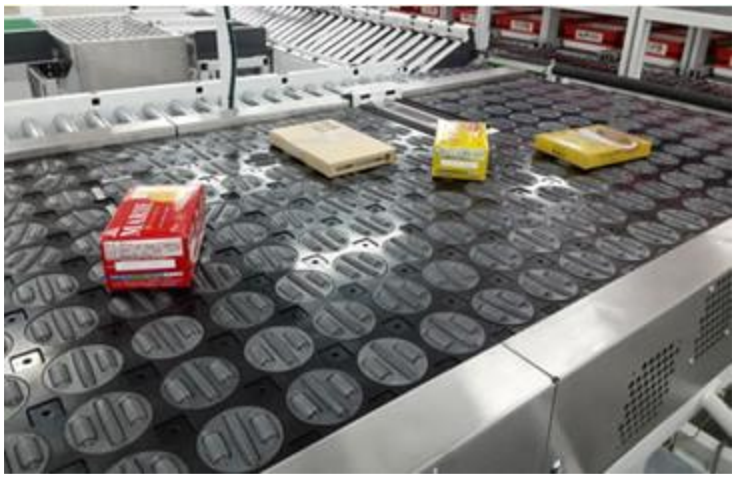

**図 1.6　マジックステージデバイス**

3. HoloTile（Walt Disney Imagineering）[4]

   HoloTile は上面の円形プレートの回転により対象物の搬送を行う。複数の円形プレートに同じ傾きを与えることで、円形プレートが同じ円周位置で搬送対象物に点接地し、回転による動力伝達方向に指向性を持たせる。これにより任意の方向への搬送を可能としている。HoloTile は人の移動手段として開発された装置であり、VR と組み合わせて人が歩く動作に合わせて歩く方向とは反対に搬送を行うことで、その場から移動させることなく VR 空間を本当に歩いているような錯覚をさせることが出来る。

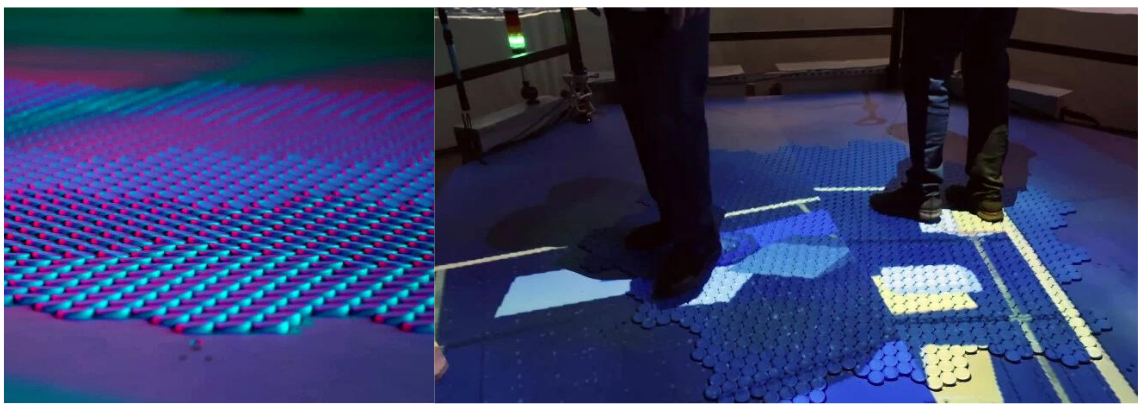

**図 1.7　HoloTile**

## 1.4. 研究目標

ユークリーターは複数の装置を連携させることで、1つの搬送システムとして運用する。しかし、これまでは装置単体としての性能向上や動作プログラムの開発が行われていた（これまでの開発状況は第2章に後述する)。そこで、本研究では装置をユニット群として使用する際に発生する空きスペースの有効活用として、太陽電池の利用によりエネルギーの時給を行うシステムの構築や、複数装置を連携させて機械自身による制御処理を行い、目的地へ対象物の搬送を可能とするシステムの構築により、自律して運転させることを目的として、制御回路の作製や制御プログラムの開発を中心に研究を行った。

# 第2章　　先行研究

本章では 2015 年からこれまでに行われていた研究をまとめる。本研究室において 1.2 で述べたシステムコンセプトから、複数の小球体の点接地による回転動力伝達により搬送を行う球体型ユークリーターと、ベルトの面接地による回転動力伝達により搬送を行うベルト型ユークリーターの 2 種類の機構開発が行われた。また、複数の小型装置の個別制御を可能とするための通信システムと制御プログラムが開発された。

## 2.1. 球体型

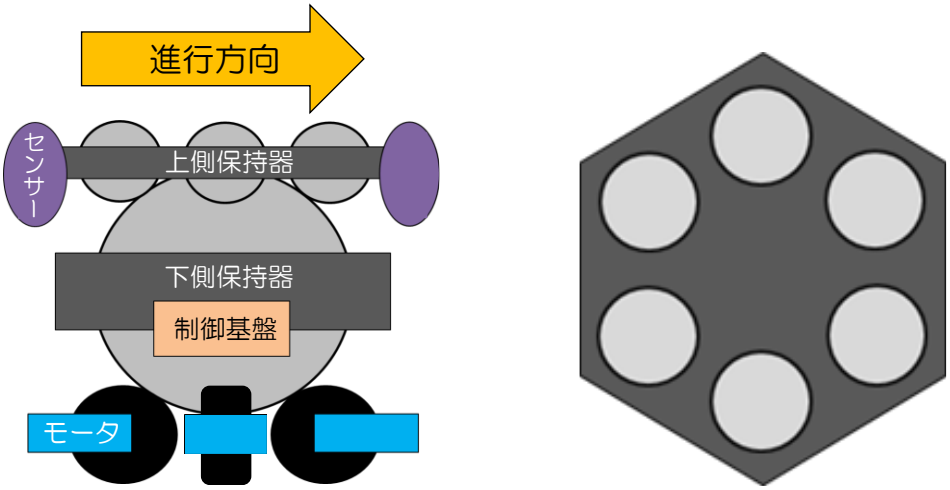


**図 2.1　基本構造の側面図および上面図**

ユークリーターの基本構造は、上段・中段・下段の 3 つのモジュールによって構成される。基本構造の側面図および上面図を図 2.1 に示す。中段にある下側保持器に取り付けられた大きな球体を、下段に設置されたモータから動力を伝えて回転させる。この時、下部球体の回転方向によって運搬物の進行方向を調整する。上段の上側保持器に取り付けられた複数の小さな球体は下部球体を通してモータの動力を分散して伝えて回転させる。当時このような原理によって、ユニット上部に乗せた対象物を運搬するという構造が立案された。

### 2.1.1. 球体型 1 号機

上記初期構想を基に床面に敷き詰められた細かな球体による全方向への搬送システムの開発を目指して試作的に作製されたのが球体型 1 号機である。球体型 1 号機の外観を図 2.2 に示す。

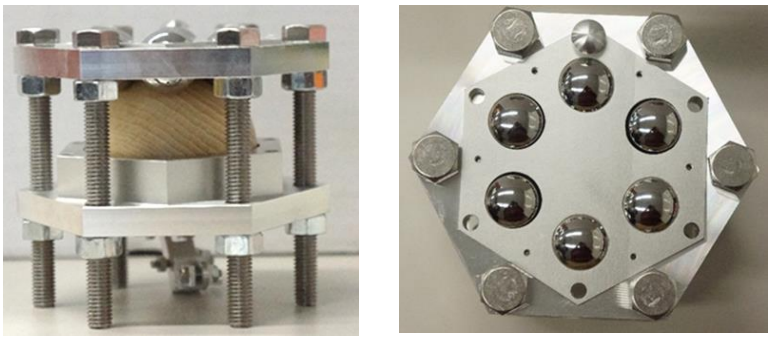

**図 2.2　球体型 1 号機の側面（左）と上面（右）**

球体型 1 号機は上面から見ると六角形のユニットで、3 つないし 4 つのモータを使用して下部の球体を制御し、上部の 6 つの球体によって運搬を行う。搬送方向は、モータの使用箇所と出力の大きさによって調整する。搬送方向の制御例を図 2.3 に示す。

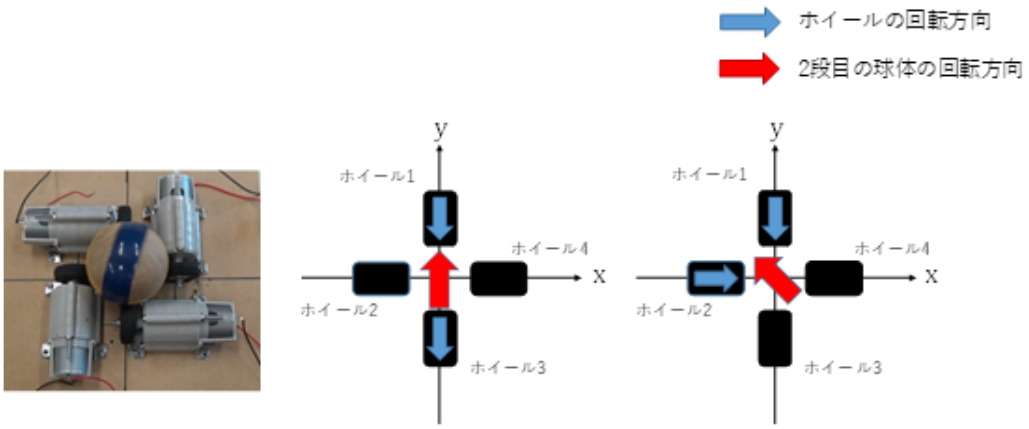


**図 2.3　球体型 1 号機の制御のイメージ**

しかし、このシステムでは対象物を運搬することはできなかった。当時対象物を運搬できなかった理由として 3 つの問題点が挙げられた。

1. 球体と保持器間の接触面積が大きすぎること
   球体とその球体を支えるための保持器間の接触面による摩擦抵抗が大きくなってしまい、モータからの動力を上面の 6 つの球体に伝えるまでの損失が大きすぎたからだと考えられた。

2. ユニット上面の余白部分が多いこと
   搬送に利用しない余白部分が多いと、運搬時に対象物がその余白の部分に接触してしまい、摩擦抵抗が生まれて運搬の妨げとなってしまうからであると考えられた。(現行システムにおける実験結果を参考にすれば2はあまり主体的な理由とは言えないかもしれない)
3. 運搬速度の制御と運搬方向の制御の 2 つの制御を、モータの使用箇所と出力の大きさでまとめて操作しようとしていたこと
   必然的にモータ出力の細かな設定が求められ、全方向に運搬を可能にするには制御パターンが極めて複雑となった。また、回転方向の一致していないホイールが一つの球体に対して回転動力を伝えようとすると、抵抗が大きくなりすぎるという不具合も起こった。

### 2.1.2. 球体型 2 号機

球体型 1 号機で見つかった問題点を改良する形で作製されたのが球体型 2 号機である。球体型 2 号機の外観を図 2.4 に示す。


**図 2.4　球体型 2 号機の全体図**

球体型 2 号機では主に 3 つの改善が施された。

1. 保持器にボールプランジャを利用
   上部の球体を支える保持器にボールプランジャをはめ込むことで、球体と保持器間の摩擦を減らす構造にしたことである。これにより、上面球体の回転がスムーズに行われるようになり、動作時のモータ動力損失が減少した。球体保持方法と使用したボールブランジャを図 2.5 示す。

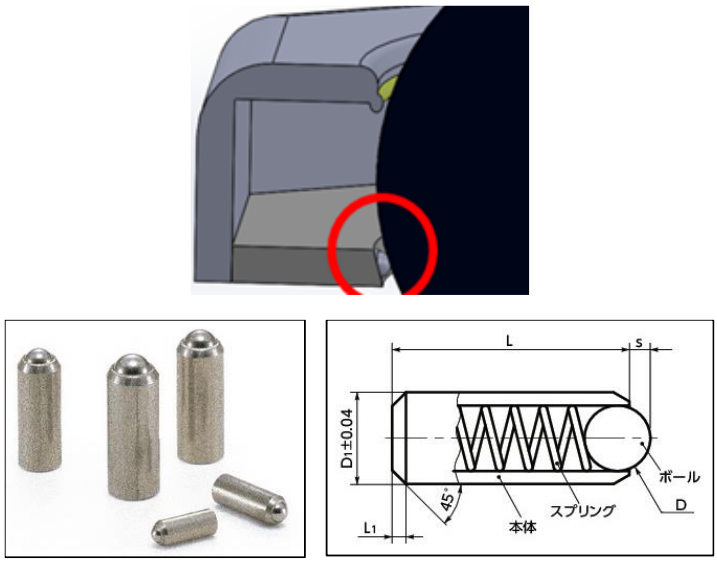


**図 2.5　球体型 2 号機の球体保持方法と使用したボールプランジャ**

2.　上面の余白部分の削減

3 段目に取り付けられている球体数を 6 つから 3 つに減らし、直径を大きくした。また、ユニット上面の形を正六角形から三角形のような変則的な六角形にすることで、ユニット上面における余白の面積を減らし、運搬物と余白部分との接触による摩擦抵抗の発生を防ぐことに努めた。

3.　使用モータ数の削減

2 段目の大きな球体の中心に軸を通し、回転するテーブルの上に乗せることで、回転制御に使用していたモータ数を、球体の回転とテーブルの回転のための 2 個に減らした。それにより、制御システムが単純化された。

動作システムは次のようになる。

A：運搬方向の決定

A1　1 段目に設置されたサーボモータを駆動させて小歯車を回転させる。

A2　小歯車の回転が大歯車に伝達される。

A3　大歯車とともに、2 段目の大きな球体を乗せたテーブルが回転し、運搬方向を決定する。

B：対象物の運搬および速度の決定

B1　DC モータを駆動させ、回転動力をベルトへ伝える。

B2　ベルトを通じて大きい球体を回転させる。

B3　2 段目の大きな球体の回転を、接地された 3 段目の 3 つの小さな球体に伝える。

B4　3 つの球体が回転し、上部に乗せた対象物の運搬を行う。

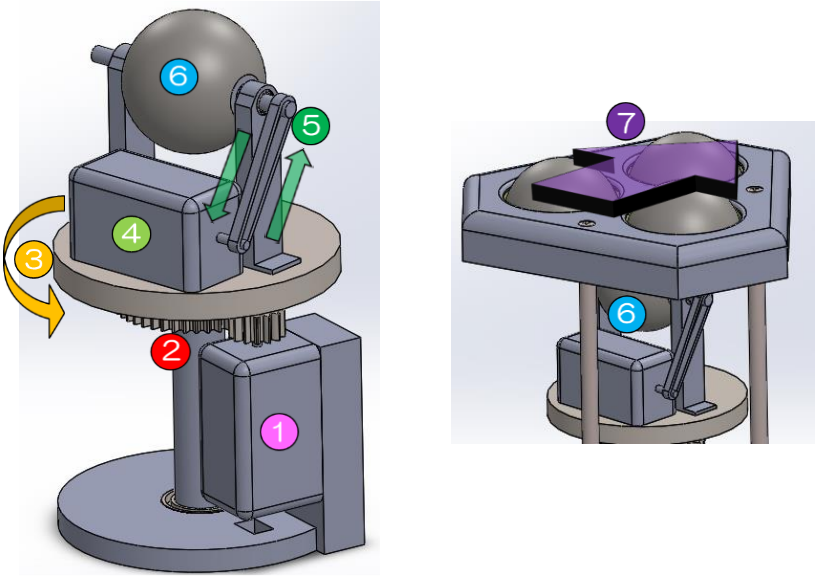


**図 2.6 球体型 2 号機の動作原理のイメージ**

球体型 2 号機では球体同士の接触により物が運搬できることが確認された。参考までに、この時提案されたユークリーターのユニット制御方法は現行システムでも採用されている。

### 2.1.3. 球体型 3 号機

次に、複数個ユニットを敷き詰めて対象搬送物の運搬を可能にするため改良を加えた新型ユニットとして、球体型 3 号機の開発が行われた。球体型 3 号機の全体図と上面図を図 2.7 に示す。

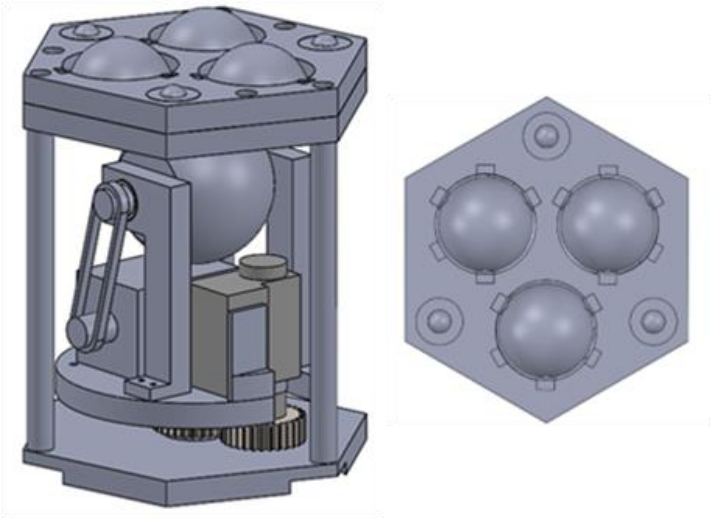

**図 2.7　球体型 3 号機の全体図と上面図**

球体型 3 号機では主に 5 つの改良が施された。

1. 小型化

   対象物を運搬する際に 1 ユニットあたりの荷重を少なくするために、球体型 2 号機では概略寸法が 200×200×300(mm)であったのに対し、球体型 3 号機では 140×140×160(mm)まで小型化をした。

2. 球体保持器の形状と球体の保持方法の変更

   球体保持器の形状は、球体型 2 号機では変則的な六角形であり、床面に敷き詰める際にできる隙間を別のユニットを用いて埋めなければならなかったが、球体型 3 号機では正六角形に変更し、1 種類のユニットのみで平面上をどこまでも敷き詰められるようにした。また、形状変化に伴い余白部分が発生してしまうため、ボールローラを用いて補った。形状変化の比較を図 2.8 に示す。

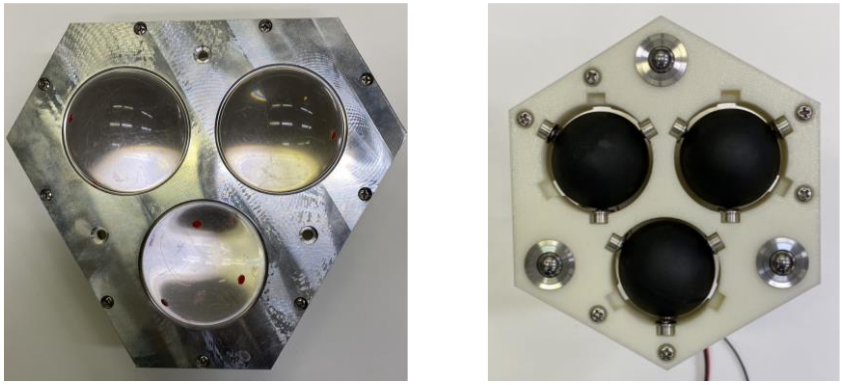

**図 2.8　上面の形状変化の比較**

3. 球体の保持方法の変更

   球体型 2 号機では図 2.5 に示したように下側のみにボールローラを設置して球体を保持していたが、モータ駆動による動力伝達時に球体が上に押し上げられることが確認された。そこで球体型 3 号機では図 2.9 のように上下から挟むように保持をする方法へ変更した。

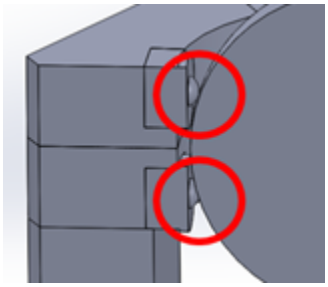

**図 2.9　球体型 3 号機の球体保持方法**

4. 上面球体の素材の変更
   上面の球体の素材をアクリルからクロロプレンゴムへ変更した。アクリル球では、2 段目の鋼球との摩擦が小さく、モータの動力の伝達損失が多かったため、摩擦の大きいゴム球へ変更した。

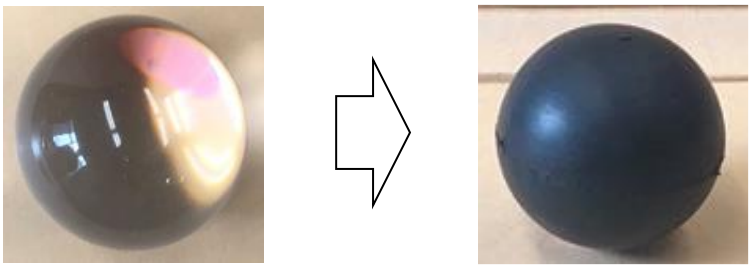

**図 2.10　アクリル球（左）とゴム球（右）**

5. サーボモータの設置位置の変更
   テーブル下にあったサーボモータの位置をテーブルの上へ変更した。テーブルと一体となったサーボモータが方向を決定する仕様に変更することで、球体型 2 号機では 2 段使用していた回転構造が 1 段に短縮され、これにより高さを約 9 ㎝小さくすることができた。

## 2.2. ベルト型

### 2.2.1. ベルト型 1 号機

球体型ユークリーターでは点接触によって搬送物への動力伝達を行っているが、損失が大きく動力伝達効率が低いという問題を抱えていた。そこで、タイミングプーリとタイミングベルトを利用することで、面接触による搬送物への動力伝達効率の大幅な改善を目的とし、球体型から派生する形で開発されたのがベルト型ユークリーターである。ベルト型ユークリーター 1 号機の外観を図 2.11 に示す。

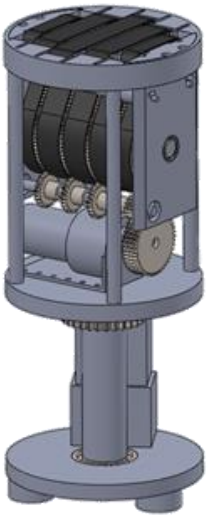

**図 2.11　ベルト型ユークリーター１号機**

しかし、当初開発されたベルト型１号機には２つの問題点が見られた。

1. 両方向に物体の搬送が不可能

   サーボモータによるユニットの 180 度の回転と搬送用動力モータの正・逆回転により全方向への搬送に対応していたが、搬送用動力モータを駆動させた際に図 2.12 に示すように搬送部に歪みが生じてしまい、特に逆回転時にはタイミングプーリ同士がずれることによりモータが空転してしまっていた。これはプーリの駆動に必要なトルクが大きすぎるため生じてしまうと考えられた。

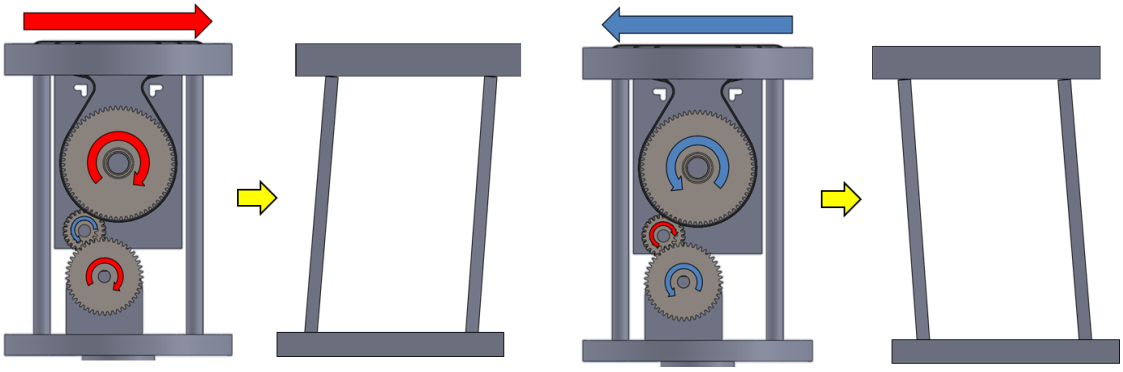

**図 2.12　正回転と逆回転による搬送部の歪み**

2. ユニット全体の振動

搬送用動力モータの駆動によりユニット全体が微振動してしまう様子が顕著に見られた。ベルト型 1 号機は直径が 110 mm なのに対し、高さは 240 mm となっており、機構全体において高さ方向への安定性が著しく低く、駆動時に微振動が発生してしまうと考えられた。勿論その振動に耐えられるほど堅牢な構造にすることも 1 つの手ではあるが、できるだけ 1 ユニットの製作コストを安価にするには構造そのものの設計仕様を見直した方が良いと考えられた。

### 2.2.2. ベルト型 2 号機

ベルト型 1 号機の問題点を改善したのがベルト型 2 号機である。ベルト型ユークリーター 2 号機の外観を図 2.13 に示す。以下ベルト型 1 号機から 2 号機の改善点を記す。

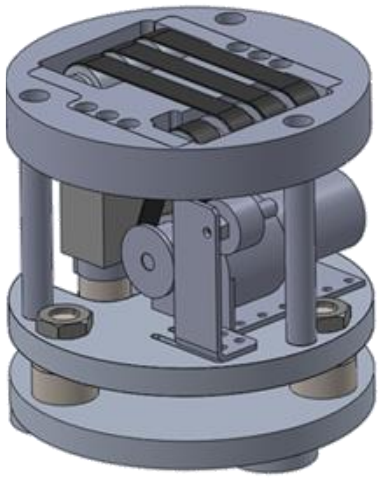

**図 2.13　ベルト型ユークリーター2 号機**

1. ベルトの駆動方式

ベルト型 1 号機では駆動プーリが中央にあるセンタードライブ方式を採用していたが、この駆動方式では高さがかさみ、構造およびコスト面からモータ駆動時にユニットの微振動発生を抑えるのは困難であると判断された。そこで駆動方式を見直し、ヘッドドライブ方式に変更することで必要プーリ数を削減し、さらにユニットの高さを抑えるなど設計仕様そのものを見直すことで微振動問題の改善に成功した。2 種類のベルトの駆動方式を図 2.14 に示す。

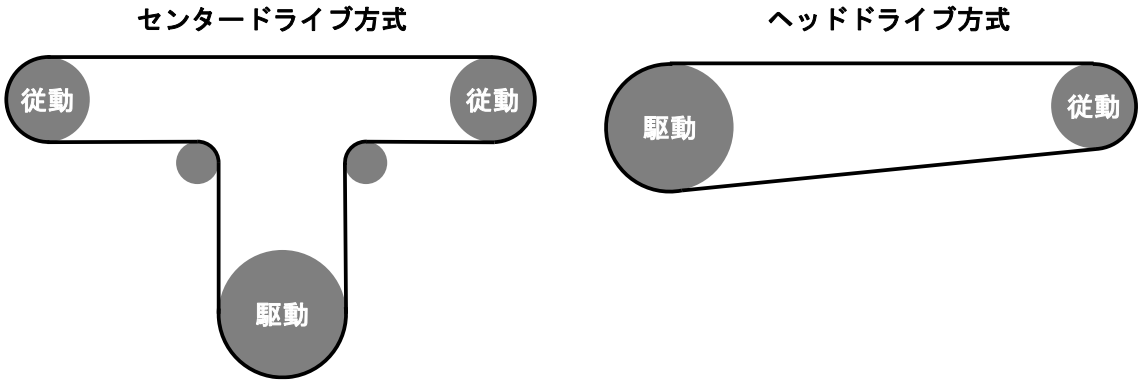


**図 2.14　2 種類のベルトの駆動方式**

2.　駆動プーリ径の縮小
　　駆動プーリの径を縮小することによってプーリの駆動に必要なトルクが減少し、構造の堅牢性を大きく改善することなく、搬送部の歪みの解消を計った。これにより正・逆回転の両回転ともに 5 kg までの搬送が可能となった。

### 2.2.3.　ベルト型 3 号機

ベルト型ユークリーターの更なる機能向上のために改良を加えたのがベルト型 3 号機である。ベルト型ユークリーター3 号機の外観を図 2.15 に示す。以下ベルト型 2 号機から 3 号機の改善点を記す。

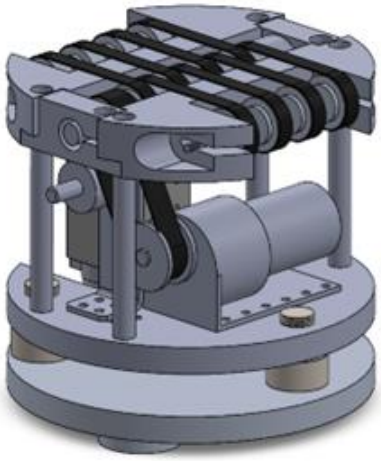

**図 2.15　ベルト型ユークリーター3 号機**

1. ベルト配置面の拡大

ベルト型 3 号機では搬送可能重量の向上のため、ユニット上面におけるベルトの配置面積の拡大が図られた。2 号機と 3 号機の上面部におけるベルト配置面積の比較図を図 2.16 に示す。この改良により、ユニット上面の空きスペース、いわゆる搬送に影響を及ぼさないスペースの縮小化が達成され、より小型な物体の搬送も可能となった。

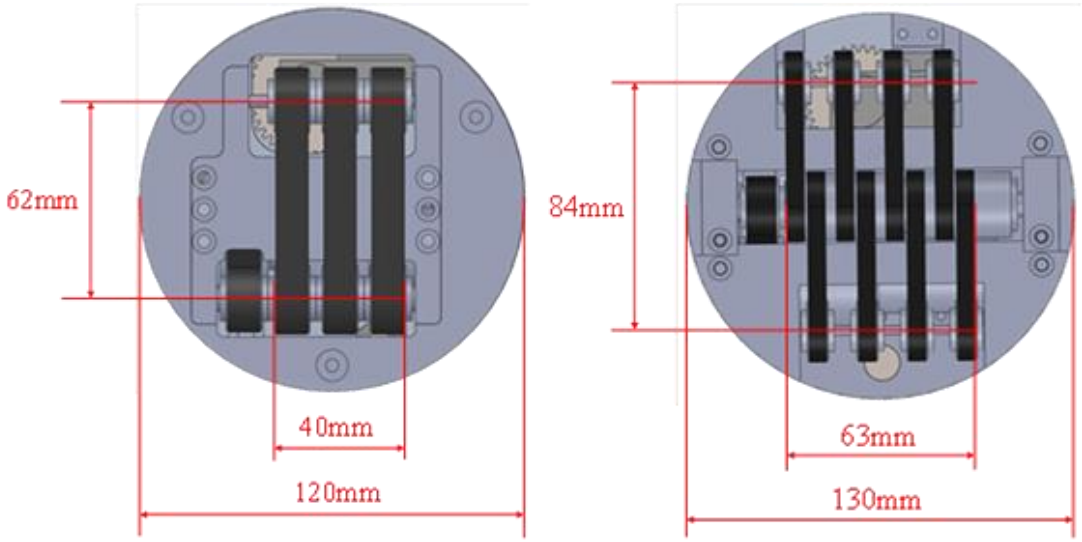


**図 2.16　2 号機と 3 号機の上面部におけるベルト配置面積**

2. 斬新な駆動プーリの配置の提案

駆動方式はヘッドドライブ方式のまま中央に駆動プーリを配置し、従動プーリを互い違いに配置するという従来手法には見られない斬新な配置が提案された。これにより、どちらの方向に搬送を行ったとしても必ずベルトに張り側と緩み側の両方が存在し、一定の搬送能力を発揮できるような構造となった。ベルト型 3 号機の搬送イメージを図 2.17 に示す。しかし、実験を行うとベルトの緩みが発生してしまい、従動プーリの位置調整を頻繁に行う必要性が生じた。これは、軽量化のためにテンションプーリを設けていないことが要因であると予想された。

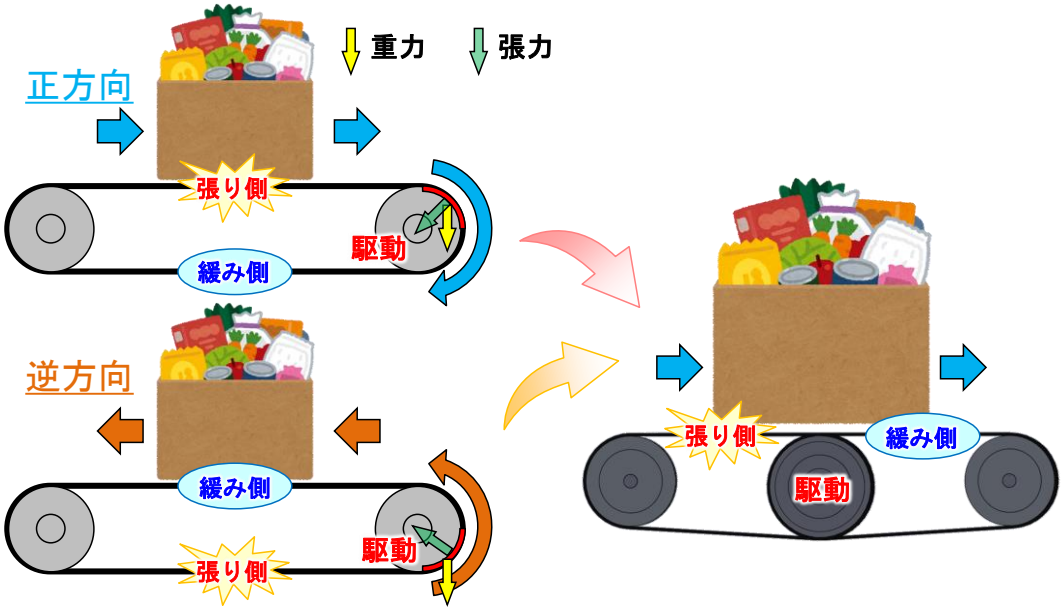


図 2.17　ベルト型 3 号機の搬送イメージ

### 2.2.4. ベルト型 4 号機

ベルト型 3 号機で発生したベルトの張力調整問題を、テンションプーリによる自動張力調整機能により改善させたのがベルト型 4 号機である。ベルト型ユークリーター4 号機の外観と上面図を図 2.18 に示す。

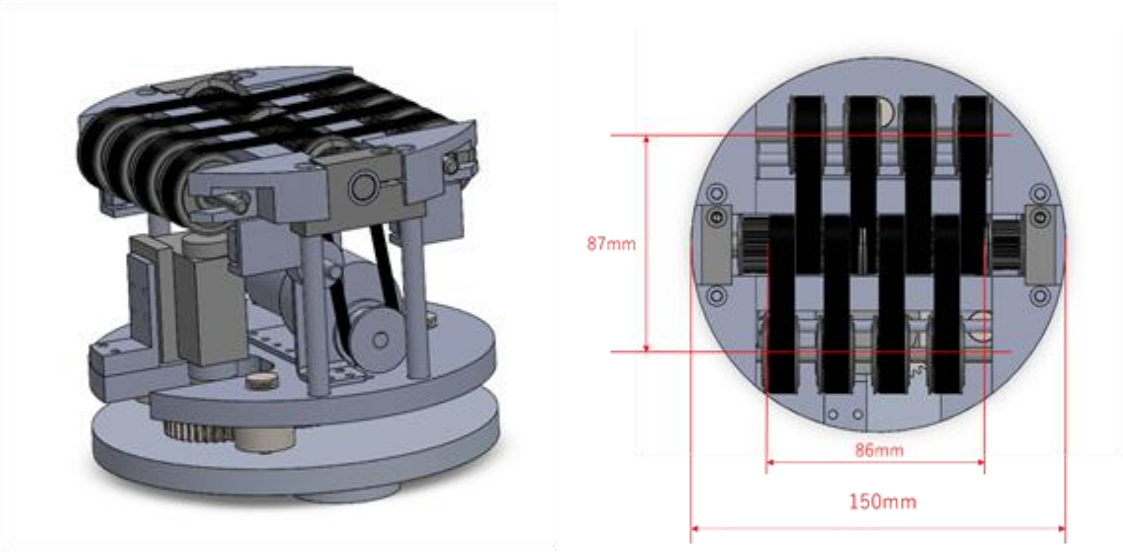


図 2.18　ベルト型ユークリーター4 号機

テンションプーリの取り付け方式はスプリング方式と重鎮方式の 2 種類から検討された。いずれの方法であっても、プーリとベルトのまき付け角が増加し、テンションプーリを用いない時よりも小さいトルクで駆動できるようになることが予想されるが、装置構造内部に存在する空き領域に余裕を持って配置でき、取り付けによる装置高を大きく変更する必要が無いことから、スプリング方式が採用された。テンションプーリの取り付け方式を図 2.19 に示す。

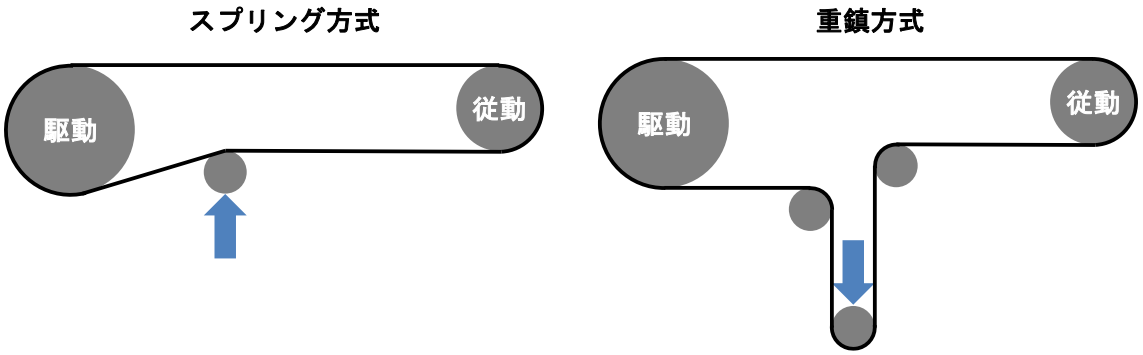


図 2.19　テンションプーリの取り付け方式

3 号機の構造ではベルトと駆動部の間に僅かな隙間しかなかったため、テンションプーリの取り付けに適したプーリ・ベルトへ変更を行い、それに伴いユニット上面のベルト占有面積の増加も図った。これらの改良の結果、搬送荷重の向上が確認された。各種ユークリーターとその他の現行搬送装置の搬送能力と仕様の比較を表 2.1 に示す。

表 2.1　各種ユークリーターとその他の現行搬送装置の搬送能力と仕様の比較

| 比較項目 | 球体型 3号機 (2020.3) | ベルト型 3号機 (2021.3) | ベルト型 4号機 (2023.2) | 実用化 必要水準 | ベルト コンベア (IHI製 SCU型) | Celluveyor (cellumation) | XPlanar (Beckhoff Automation) |
|---|---|---|---|---|---|---|---|
| ①搬送速度　(m/min) | ~3.0 | 10~15 | 11~13 | 20 | 3~25 | ≈2 | ≈2 |
| ②搬送荷重　(kg/m) | ~40 | ~60 | ＞~90 | 100 | 150 | ≈100 | ≈15 |
| ②'搬送荷重　($kg/m^2$) | ~330 | ~620 | ＞~970 | ~1000 | ー | ~1800 | ~76 |
| ③搬送用筐体必要性 | 無 | 無 | 無 | ー | 無 | 無 | 有 |
| ④ユニット間連動性 | △ | △ | △ | ◎ | ◎ | ○ | ○ |
| ⑤輸送可能次元数 | 2 | 2 | 2 | 1 | 1 | 2 | 2 |
| ⑥各ユニット独立駆動 | ○ | ○ | ○ | × | × | × | ○ |
| ⑦電源自律分散化 | × | × | × | × | × | × | × |
| ⑧自立運転 | × | × | × | × | × | × | × |

## 2.3. 通信制御

ユークリーターは数百から数万台のユニットを敷き詰めて搬送対象物の搬送を行うシステムであり、搬送時には必要な箇所のみの機器を選択し、搬送対象物の大きさに伴った複数台を連動させ運転することで搬送工程を制御する。そのため、ユニット間の通信制御にはそれほどの数の同時通信が可能であり、機器が密集した状況下でも安定した通信が行える通信形態を選択しなければならない。以下先行研究においてどのような構想のもと開発が行われていたかまとめる。

### 2.3.1. 無線通信

物理的な配線を必要とせず接続台数の変更や機器の配置の変更が容易であり、また、必要箇所のみへの通信による運用がしやすいといった考えから、無線通信による制御が検討された。そこでまず無線通信用モジュール ESP8266 を搭載したユニット制御用回路基板が作製された。作製した回路基板を図 2.20 に示す。

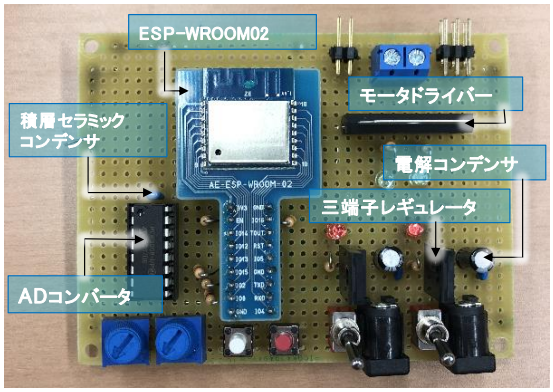


**図 2.20　無線用ユニット制御回路基板**

通信プロトコルには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を受信する通信方法である UDP 通信（User Datagram Protocol 通信）が、ユークリーターのシステムに最も適しているのではないかと想定され、この通信形態を導入した制御が試みられた。UDP 通信による通信イメージを図 2.21 に示す。

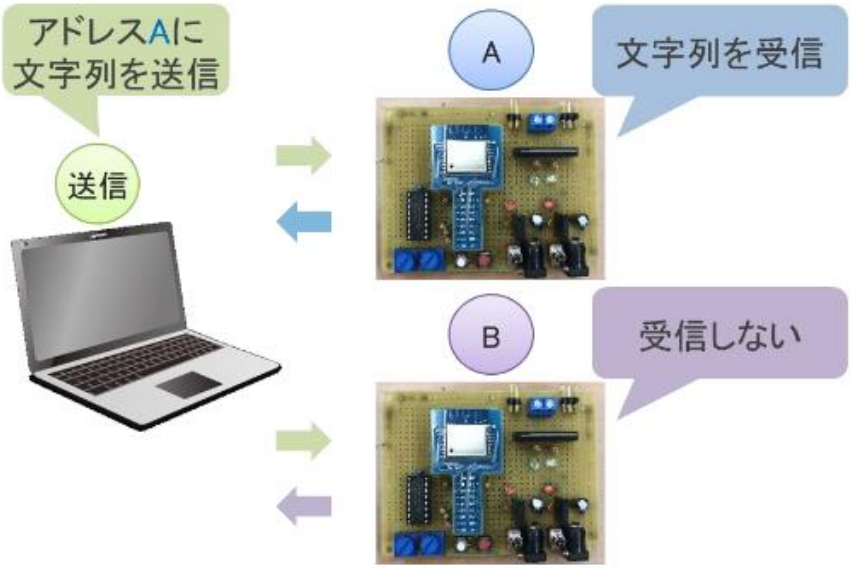


**図 2.21　PC と複数制御回路の UDP 通信イメージ**

実際に Wi-Fi を用い通信動作実験を行ったところ、PC と制御回路の一対一での通信では正常に送受信が行われ、無線通信でのユニットの搬送動作が可能なことが確認された。しかし、制御回路を増やし、一対多の通信制御を行おうとすると、ユニットの不安定な動作が見られた。当時この現象はモータによるノイズが通信の妨げをしていると考えられ、数千台単位での使用を考慮しているユークリーターでは無線による通信制御は困難だと判断された。しかしながら、本手段は将来的には有望かもしれないので改めて見直す価値はある。

### 2.3.2. 有線通信制御（シリアル通信）

次に、大量の機器によるノイズが発生しても確実な動作を可能とする方法として有線通信による通信制御が考えられた。PC から有線による通信を行う方法にはシリアル通信とパラレル通信の 2 種があり、配線数が少なく済みクロックのずれが生じないシリアル通信が選択された。シリアル通信は一対一による通信であるため、単体ユニットとの通信であれば PC とユニットを配線すれば可能であったが、複数ユニットとの通信を行うためには分配器を PC とユニットの間に接続し、PC からのデータを各ユニットに分配して送信する必要がある。また、分配器の接続可能台数を超えるユニット数と通信したい場合には分配器からさらに分配して補えば良い。シリアル通信によるユニット制御イメージを図 2.22 に示す。

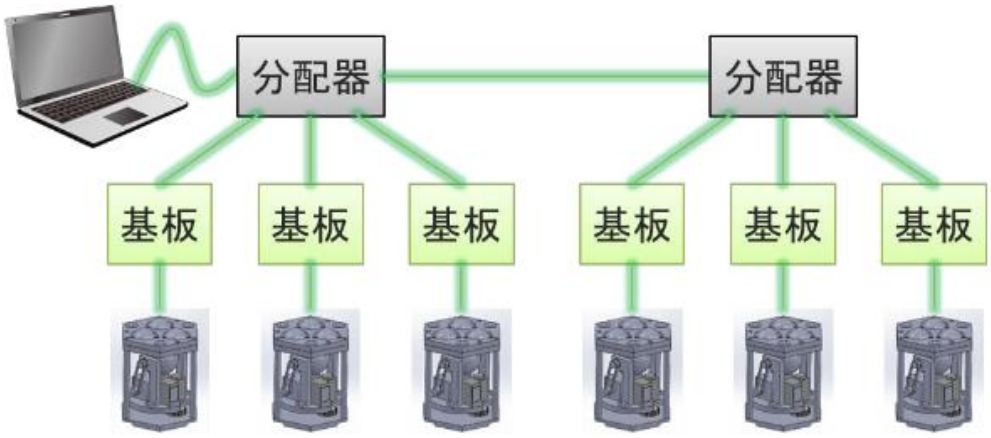


**図 2.22 シリアル通信を用いたユニット制御イメージ**

有線通信用のユニット制御回路基板を作製する際、制御用マイコンとして Arduino NANO が採用された。この通信形態では基本 1 対 1 の通信形態であることから、ある 1 ユニットにデータを送信し、受信したユニットが自身のアドレス宛てのデータか判断する。合っていればユニットの動作を、違っていたら受信データを送り返し次のユニットに同じデータを送信する。この繰り返しにより指定のユニットのみを動作させる通信プロセスを構築した。しかし実際に動作実験を行ったところ、ユニットが接続された順番にしか制御ができず、下流のユニットは動作させるのに大きな遅れが生じることが分かった。他にも分配器を用いて通信台数を増やしているため、一部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も発覚した。これらのことからシリアル通信は複数制御には向いていないことが判断された。

### 2.3.3. 有線通信制御（CAN 通信）

次に、車の制御にも利用される CAN 通信による制御が考えられた。CAN（Controller Area Network）通信とは複数の CAN デバイスを使用し、低コストかつ耐ノイズ性に優れたネットワークプロトコルのことであり、主に自動車用の通信ネットワークとして用いられている。通信線のことをバス、ネットワークに接続する通信機器（電子ユニット（ECU）など）のことをノードといい、複数のノードを一本のバスで接続して通信を行うバス型トポロジーという接続形態を採用している。通信の概念を図 2.23 に示す。


**図 2.23　CAN 通信の接続形態**

バス型トポロジーではノードの追加によるネットワークの再構築が容易で、ネットワーク設計が行いやすいという特徴がある。そのため、ユークリーターを設置する際にユニットの個数や配置の変更が行いやすいという利点がある。また、ネットワーク内の通信開始タイミングにおいてノードに優劣がないマルチマスター方式を採用している。そのため、一部のノードに故障が生じて通信に問題が生まれた場合でも、その他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ないので、故障箇所のみの取り換えが可能でありメンテナンス性においても優れている。

CAN 通信を用いた制御方法は主に 2 種類ある。1 つ目は、多数のユニットを 1 つの CAN 通信対応回路で制御する方法である。通信制御形態のイメージを図 2.24 に示す。この方法では、制御するユニットに対して少ない基板数で済むため、ユニットに搭載する回路の設計が比較的簡単になる反面、1 つの CAN 通信対応回路の故障により接続されたすべてのユニットが動作しなくなるという欠点がある。さらに、センサなどの部品を追加する場合、複数のユニットに接続する部品のすべてを 1 つの CAN 通信対応回路に納めなくてはならないため、この方法ではシステム全体を設計する際に複雑性が高くなり汎用性が低くなる。

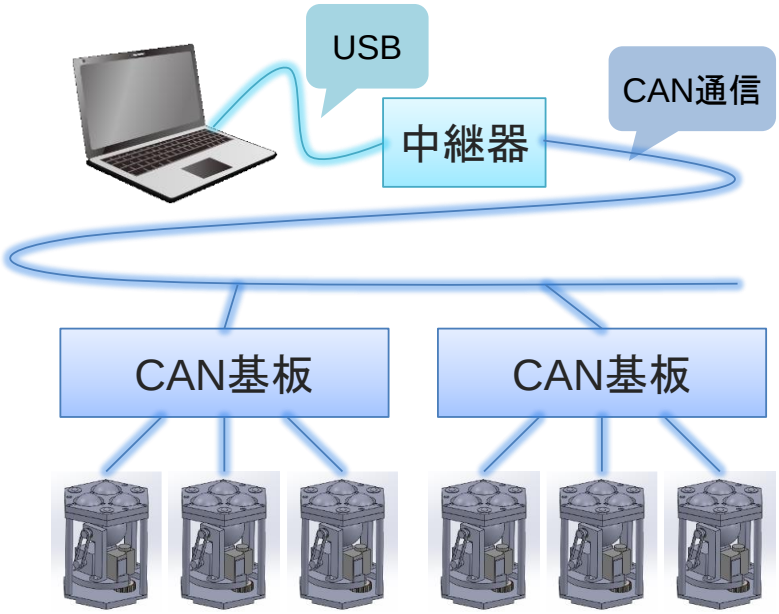


図 2.24　多数のユニットを 1 つの基板で制御する CAN 通信の概念

2 つ目は、1 ユニットにつき 1 つの CAN 通信対応回路もしくは素子を設けて制御をする方法である。通信制御形態のイメージを図 2.25 に示す。この方法では、ユニット数が増えれば増えるほどその分の回路もしくは素子を同時に準備しなければならないという欠点はあるが、ユニットを制御する基板に一緒に搭載することもでき、万が一 CAN 通信対応の回路もしくは素子が故障しても停止するユニットは 1 つだけであり、交換も容易である。また、センサなどの部品の追加への対応もしやすい。これらのことから、ユークリーターの仕様を考慮するとこの方法が適していると考えられたため、CAN 通信が採用された。

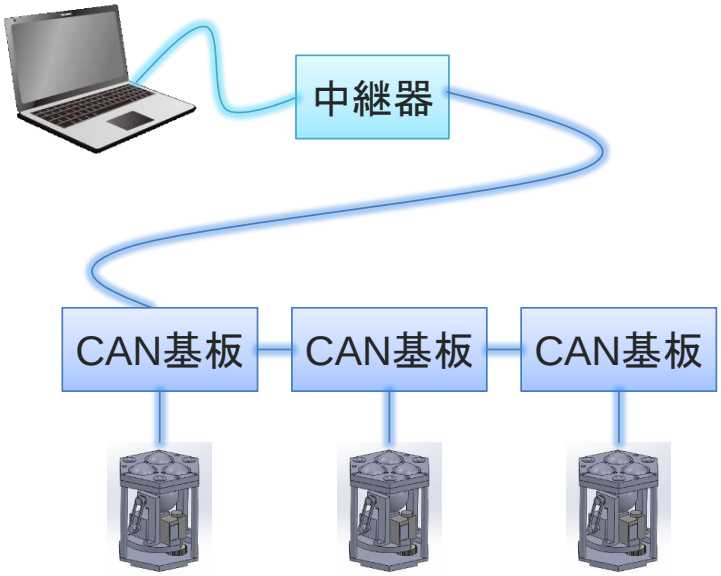


図 2.25　1 つのユニットを 1 つの基板で制御する CAN 通信の概念

CAN 通信対応回路と各ユニットの動作を制御する制御用基板を一つの基板にまとめ、PC とのシリアル通信と全ユニットとの CAN 通信を行う中継器基板を作製し、動作実験が行われた。複数台の制御を行っても、Wi-Fi による無線通信（2.3.2）の際に見られたモータなどのノイズによる動作不良も見られず、指定したユニットの制御が行えたことから CAN 通信による通信形態がユークリーターに適していると判断された。

### 2.3.4. ユニットのアドレス設定と制御方法

ユークリーターは隙間なく敷き詰めを行うためにユニットの外形は正六角形状をしている。しかし、敷き詰めた各ユニットにアドレス（位置情報）を与える際に、我々が一般に用いる直交座標系ではほとんどのユニットの座標値が整数にならないという不都合が生じる。これは計算機上ではたいした問題ではないが、人が荷物の搬送を操作する際に直感的に位置を決定しにくくなることが懸念される。直交座標系でのアドレス割り振り例（各ユニットの中心座標）と問題点を図 2.26 に示す。

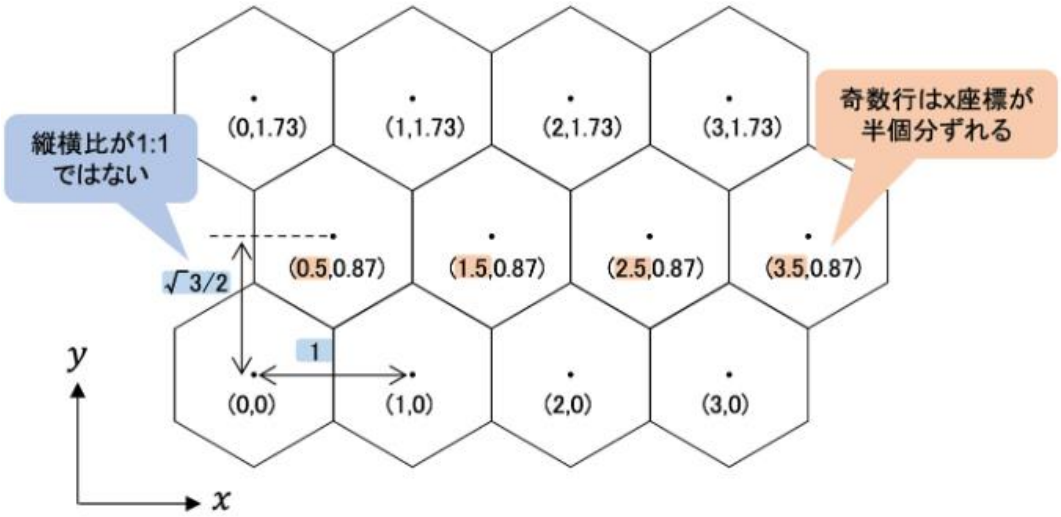


**図 2.26　直交座標系でのアドレス割り振り**

そこで直交座標系の代わりに 60°斜交座標系を用いることで、座標値が整数値として扱うことができ、ユニットの配置と座標が直感的に結びつきやすいものにした。60°斜交座標系でのアドレス割り振り例（各ユニットの中心座標）と長所を図 2.27 に示す。

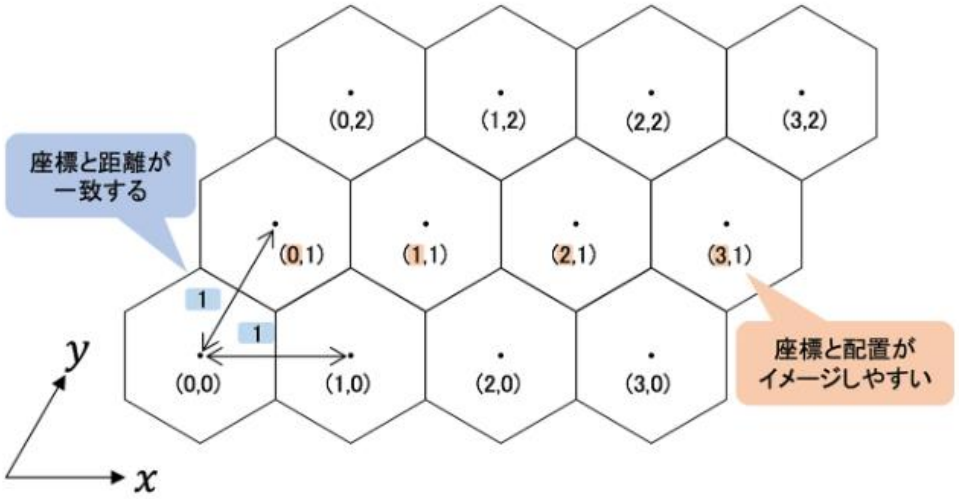


**図 2.27　60°斜交座標系でのアドレスの割り振り**

Python を利用して、PC をコントローラとしてユニットを動作させるためのプログラムを作成された。グラフの描画には matplotlib というモジュールを利用したが、このモジュールには斜交座標系の設定が実装されていなかったため、有志によりインターネット上で公開されていたモジュールの一部を利用し[20]、my_skewt.py として斜交座標系を設定するモジュールが作成された。

PC からの各ユニットの操作方法は次のようになる。

1. システム外観を表す六角形の配列に対して原点と座標を割り振り、PC の操作画面上に表わす。六角形は 1 つのユニット外形に担当する。
2. マウスクリックによって目的地点の座標を決定する。
3. マウスカーソルの位置座標を常に読み取り、その座標を現在搬送対象物がある位置として認識する。その位置から目的地点までを直線で結び、その直線の傾きをサーボモータの駆動角として計算する。
4. マウスカーソルの位置が六角形ユニット外形の内側にあるとき、その六角形に対応するユニットアドレスに向けサーボモータの駆動角と DC モータ出力を送信し、対象ユニットを動作させる。
5. マウスカーソルの位置が六角形ユニット外形の内側から外側へ変化したとき、動作させていたユニットに搬送方向の初期化と DC モータ停止命令を送信することでホームポジションへと戻す。

以上の方法で 1 台ずつ搬送動作を繰り返すことで搬送対象物を目的の地点まで搬送する。操作画面と動作実験の様子を図 2.28 に示す。

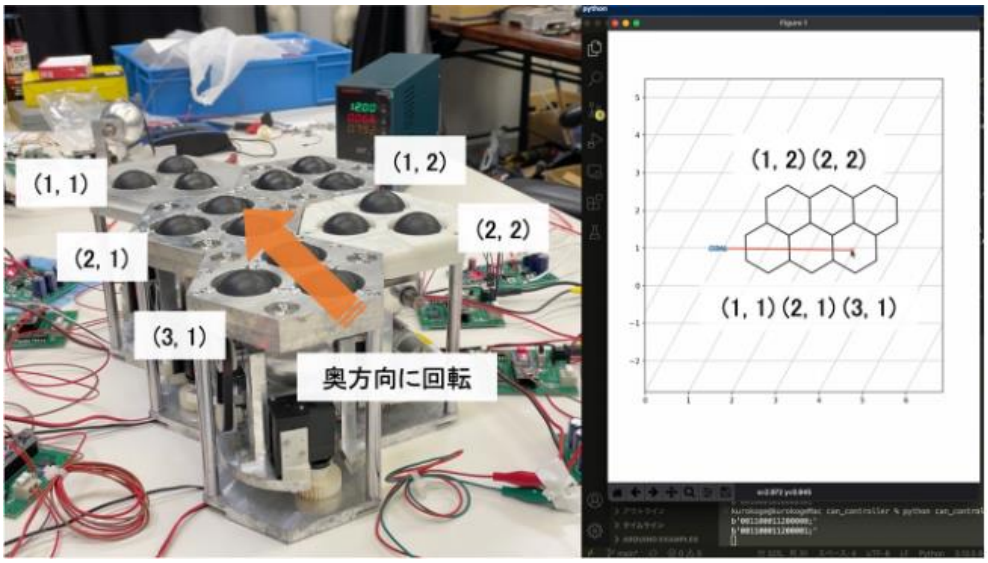


**図 2.28　操作画面と動作実験の様子**

CAN 通信コントローラの動作実験の動画
https://youtu.be/6iNNz-Ny2OA

# 第3章　太陽光発電システムの開発

## 3.1. 空きスペースの利用

球体型ユークリーターを敷き詰めた際に、図 3.1 を見ると分かるが約 70×55 $mm^2$ の搬送には使用しない空きスペースが存在する。2.1.3 節に記載したように、このスペースにより搬送効率が下がるのではないかという懸念から、スムーズな搬送を行うため、従来の球体型 3 号機にはボールローラが取り付けられていた。搬送実験を行ったところボールローラの有無による搬送効率の変化は顕著には確認されなかった。

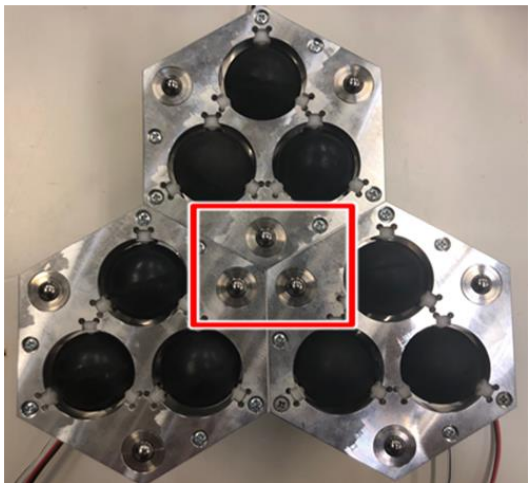

**図 3.1　ユニットを敷き詰めた際に生じる空きスペース**

しかしながら、この空きスペースを放置しておくにはもったいない。ところで、ユークリーターは外部電源に頼らず自立的な運用を目標としている。そこで太陽光パネルを上記空きスペースに設置することで有効活用し、ユークリーターへの機能付与を図る。

太陽から放出されているエネルギーは約 1370 $W/m^2$ であり、上記の空きスペース（0.00385 $m^2$）で受けられるエネルギーは約 5.27 W である。太陽光発電におけるエネルギー変換効率は 15 %～20 %程度とされているため、9 時から 16 時までの 7 時間充電を行うものとすると、1 日の発電量は約 20 kJ と考えられる。球体型ユークリーターの消費電力は搬送対象物の重量によって変動するが、例えば 2.5 kg の搬送対象物の搬送する場合に 1 台のユニットあたり 2.5 W の仕事が必要であることが過去実験からわかっている[15]。したがって、1 空きスペースから得られる発電量で球体型ユークリーターを 2 時間程度の稼働が可能となる。ユークリーターはその仕様上、必要な時のみ稼働をさせることを想定しているため、太陽電池の搭載によりユークリーターの消費電力を十分に賄うことができると考えられる。

## 3.2. 鉛蓄電池への充電を目標とした回路設計

まずは鉛蓄電池の充電プロセスを確立させるために、安定化電源から鉛蓄電池への昇降圧を行い適切な電力へ変換する充電用安定化回路の作製から試みた。作製した充電用安定化回路の写真と回路図を図 3.2 に示す。回路に使用した電子部品は、Arduino NANO、LCD、SC8721（dc-dc コンバータ）、オペアンプ 662、SD カードソケット、半固定ボリューム（10 kΩ）、カーボン抵抗器（1 kΩ、3 kΩ、5 kΩ、0.1 kΩ、9.1 kΩ）、2.1 mm 標準 DC ジャック（4UCON 製）、ターミナルブロック（Alphaplus 製）、積層セラミックコンデンサ（0.1 μF）（村田製作所製）、低電圧ショットキーダイオード（SBM1045VSS）（PANJIT 製）、低損失三端子レギュレータ（TA48M05F）（TOSHIBA 製）、電解コンデンサ（35PX47MEFC5X11）（Rubycon 製）である。

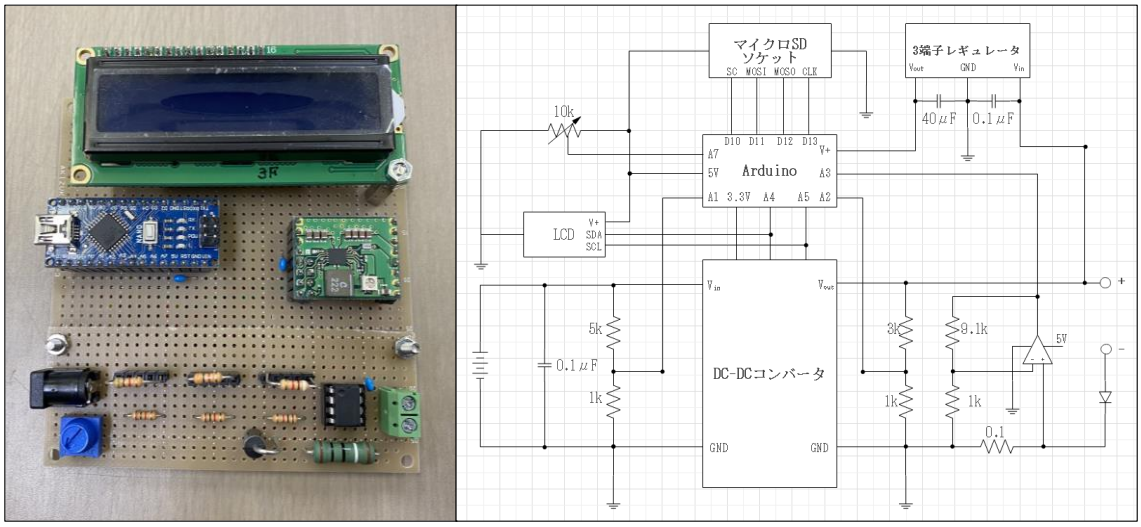


**図 3.2　充電用安定化回路の写真と回路図**

鉛蓄電池は充電率によってインピーダンスが異なり、充電率が低いときはインピーダンスが小さく、充電率が高くなるにつれてインピーダンスは大きくなる。よって充電時の最大充電電圧と最大充電電流の定格を超えない最大電力に調整し鉛蓄電池に出力することで最大効率充電を図る。鉛蓄電池はユークリーターの DC モータを最高効率で運用するため 12 V で放電を行える、完全密封型鉛蓄電池 WP1.2-12 を採用した。充電電圧は 14.4～15 V、最大充電電流は 0.36 A、定格容量は 24 Ah である。使用した鉛蓄電池の写真と理想的な充電電力図を図 3.3 に示す。理想的な鉛蓄電池の充電電力は、前半は最大定格電流による定電流過程であり充電電圧は上がり続け、後半は最大定格電圧による定電圧過程であり充電電流は下がり続ける。

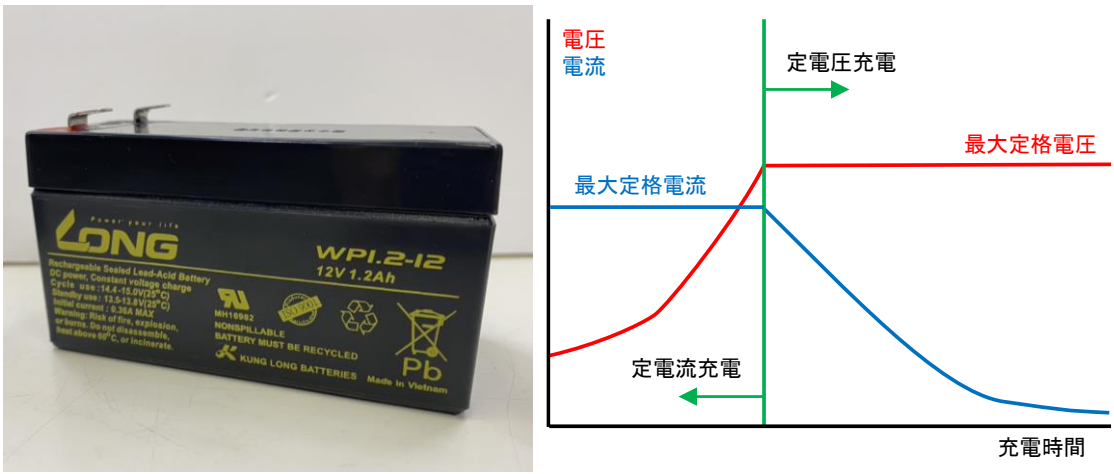


**図 3.3　完全密封型鉛蓄電池 WP1.2-12 と理想的な充電電力図**

## 3.3. 安定化電源を用いた充放電の確認

スイッチング型昇降圧 dc-dc コンバータである SC8721 を使用し、5 V から 22 V の間を 0.02 V 刻みで出力電圧の制御を行うこととした。また、入力電圧と出力電圧・電流値を Arduino NANO で取得し SD カードに記録をするようにした。入力電流の測定も予定していたが、差動オペアンプによる電圧の増幅が出来なかったため、入力電流の測定が行えなかった。Arduino NANO の電源は充電を行う鉛蓄電池に接続をすることで供給を行う。その際に鉛蓄電池は 12 V 出力なので、Arduino NANO の電源に合わせた電圧にするために 3 端子レギュレータにより 5 V に降圧し、電解コンデンサを経由することで平滑化し、5 V ピンを通じて Arduino NANO へ電源入力をした。

充電プロセスは、初期設定出力電圧を 11 V とし鉛蓄電池へ電力供給を開始する。0.25 秒ごとに実際の鉛蓄電池への出力電圧と出力電流を読み取りながら常に 0.02 V ずつ昇圧を行う。出力電圧が 14.4 V を超えたら 0.02 V 降圧、または出力電流が 0.36 A を超えたら 0.04 V 降圧を繰り返すことで、鉛蓄電池の許容最大充電電力での充電を図る。充電電圧は定格では 14.4 V～15 V だが、設定上限電圧を超えるまで昇圧し続けるという、超えることを前提として電圧制御を行う制御プロセスのため、設定上限電圧を定格電圧の下限である 14.4 V に設定した。充電電圧が 13.5 V 以上で充電電流が 0.1 A を下回ったときに充電完了として充電プロセスを終了し、鉛蓄電池への出力電流が 0 A となるようにフロート電圧をかけ続けて満充電状態を維持する。実際に電圧値を 18 V に設定した安定化電源を、作製した充電用安定化装置に接続し、鉛蓄電池の電圧と電流値を読みながら出力電圧の昇降圧を行い、充電完了判定を行うまでの鉛蓄電池へ充電を行ったときの記録を図 3.4 に示す。

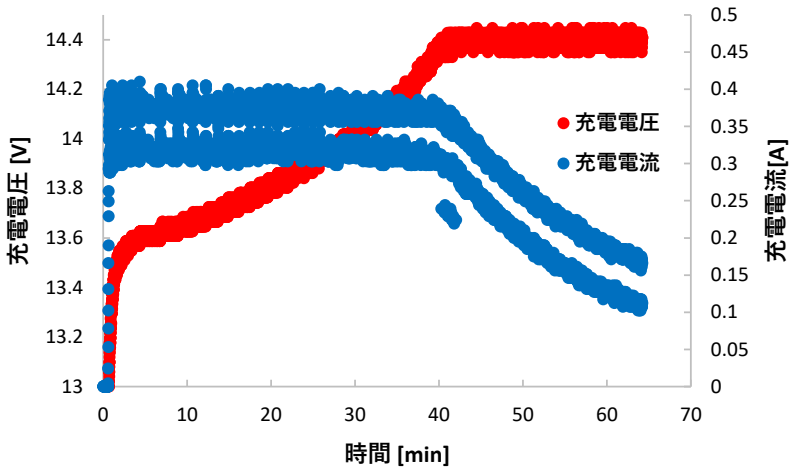


**図 3.4　電圧値を 18 V に設定した安定化電源からの充電記録**

0.25 秒毎に 0.02 V の昇降圧を行うため、それに準じて出力電流は段階的な値を取り、定電流・定電圧制御の時には二分化した電流値として記録された。しかし、2.5 秒ずつ電圧・電流値を平均化してグラフにすると図 3.5 のようになり、 理想的な充電電力図と同じグラフを描くことから、安定化電源から鉛蓄電池へは最大効率での充電動作が行われていることが確認された。

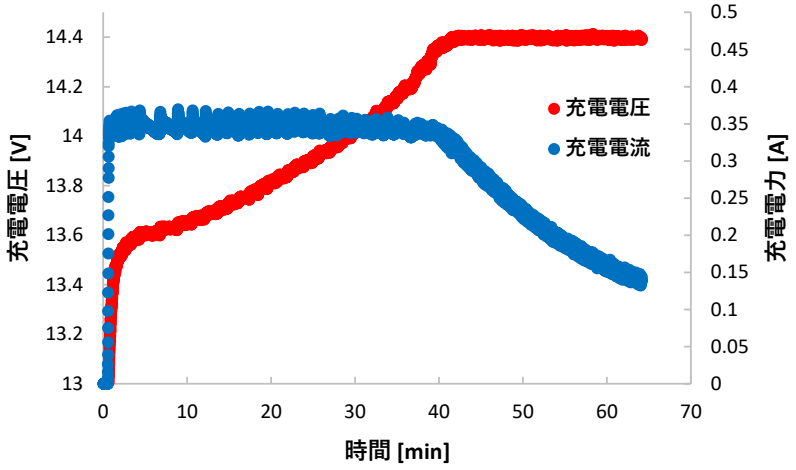


**図 3.5　2.5 秒で平均化した電圧値を 18 V に設定した安定化電源からの充電記録**

また充電システムに併せて放電システムも作成した。鉛蓄電池を放電し切ると電池内の負極板表面に硫酸鉛の結晶が生じ（サルフェーション）、電気容量の低下が起こってしまうため、放電出力を制御しなければ鉛蓄電池の寿命が縮んでしまう。そこで、放電プロセスでは全て放電し切らないようなプログラムを作成するように注意する必要がある。よって、常に

放電電流が 1 A になるように出力電圧の昇降圧制御を行い、放電電圧が 11.5 V を下回ったら放電を終了するような仕様とした。

## 3.4. 太陽光による充電

安定化電源から鉛蓄電池への充電および放電を行えることが確認されたので、次は実際に太陽光パネルを用いて鉛蓄電池への充電を試みた。鉛蓄電池の充電には 5.8 W（14.4 V ×0.4 A）程度の出力があれば充電できることを先の実験ですでに確認していたので、10 W の出力を有する太陽光パネルを購入して使用した。太陽光パネルの性能表を表 3.1 に示す。

**表 3.1**

| | |
|---|---|
| 面積 | 390×292 mm |
| 開放電圧 | 21 V |
| 短絡電流 | 0.65 A |
| 最大電力 | 10 W |
| 最大電圧 | 18 V |
| 最大電流 | 0.58 A |

安定化電源から鉛蓄電池への充電実験の際に用いたものと同じ鉛蓄電池とプログラムを使用して、2022/11/10 の 11 時頃に雲に太陽光が遮られることのない晴天下で実験を行った。まず充電開始当初、安定化電源を用いた場合のような安定した充電挙動が見られなかったが、しばらくすると電流値が安定した。しかし、さらにしばらくすると電流・電圧値ともに不安定になったので充電を中断することとした。そのときの充電記録を図 3.6 に示す。図 3.4 と比べると、定電流充電になるまでの昇圧過程において電流値が 0~1 A の間で大きく振れていることが確認される。その後電流値は安定し、0.36 A 付近での定電流充電が行われており、安定化電源を用いて鉛蓄電池への充電を行った際の充電挙動とほとんど同じだった。しかし、10 分を超えたあたりから突然電流値が 0 A になり、昇圧処理を行うものの実際の充電電圧は上がらず充電不可となってしまった。

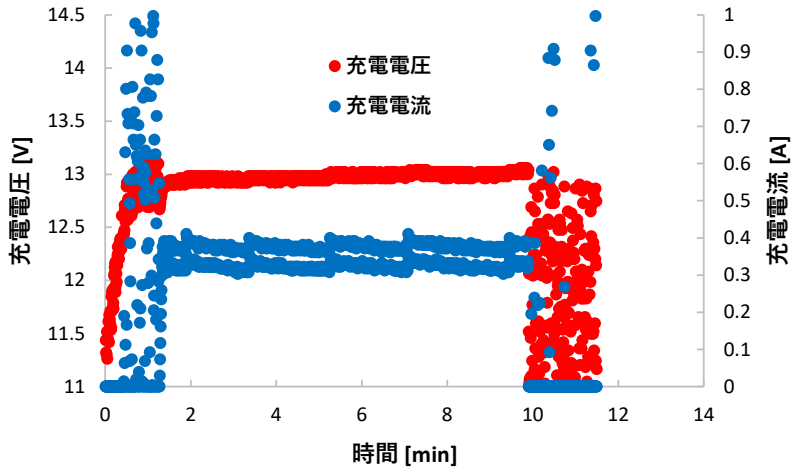


**図 3.6　太陽光発電記録 1**

このような結果となったのは、太陽光パネルから供給される電力量の不足、もしくは回路の不備やプログラムの不備による鉛蓄電池に供給される電力量の不足が疑われる。しかしながら、安定化電源を用いた際の充電が上手く行われていることから、少なくとも回路そのものに不備はないと思われ、太陽光パネルから供給電力量の不足によって鉛蓄電池への充電が失敗したものと予想し、プログラムの修正方針を決定するため太陽光パネルの供給電力特性を調査することにした。

## 3.5. 太陽光パネルの電力出力特性調査

鉛蓄電池では充電率によりインピーダンスが変化してしまうことから、太陽光パネルがどのような特性を有しているか調査することが難しい。そこで、鉛蓄電池の代わりに 10 Ωの抵抗器を用意して昇降圧回路の出力側に接続し、設定出力電圧 5 V から 2.5 秒毎に 0.02 V ずつ昇圧させながら抵抗器への出力電力を記録することとした。その際の出力電圧と出力電流の値を図 3.7 に示す。実験は 2022/11/21 の 13 時頃に雲に太陽光が遮られることのない晴天下で行った。

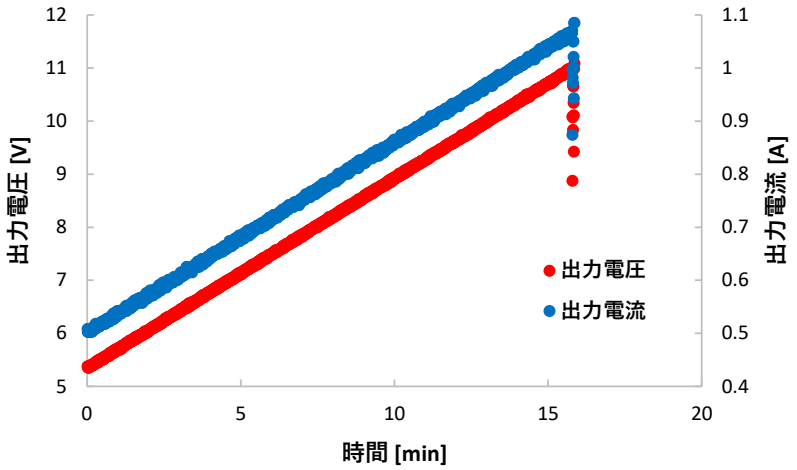


図 3.7　太陽光パネルから抵抗器への昇降圧実験記録

本実験結果から、今回採用した太陽光パネルでは電圧 10.98 V、電流 1.07 A の約 11.75 W の出力まで供給可能なことが分かった。出力電力とパネル電圧の関係をグラフにまとめたものを図 3.8 に示す。パネル電圧は出力電力の増加に伴って減圧していき、12 W に近づくにつれ減少幅の増大が確認され、電圧降下が顕著になることがわかった。

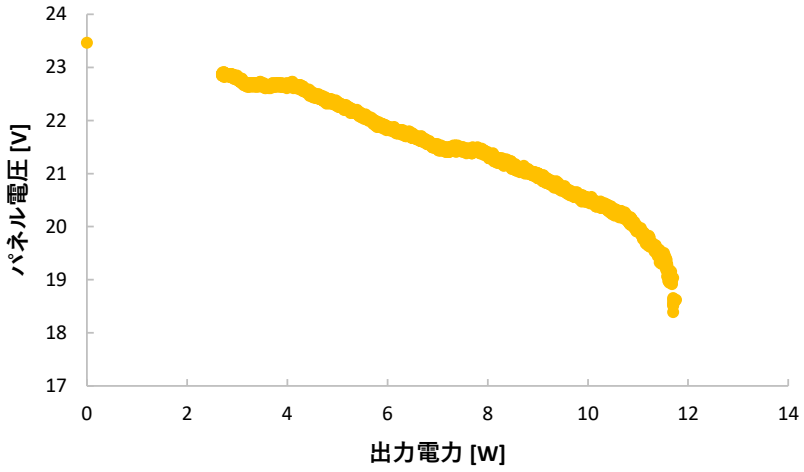


図 3.8　出力電圧とパネル電圧の関係

ところで、抵抗体と半導体である太陽電池では電流電圧特性が異なる。それぞれの電流-電圧特性は図 3.9 に示すように、単なる抵抗が直線になるのに対し、太陽電池の場合は右肩が膨らむような IV 曲線となる。最大電力点を境に、電圧に対して電流の変化量が大きい領域と小さい領域に分かれる。

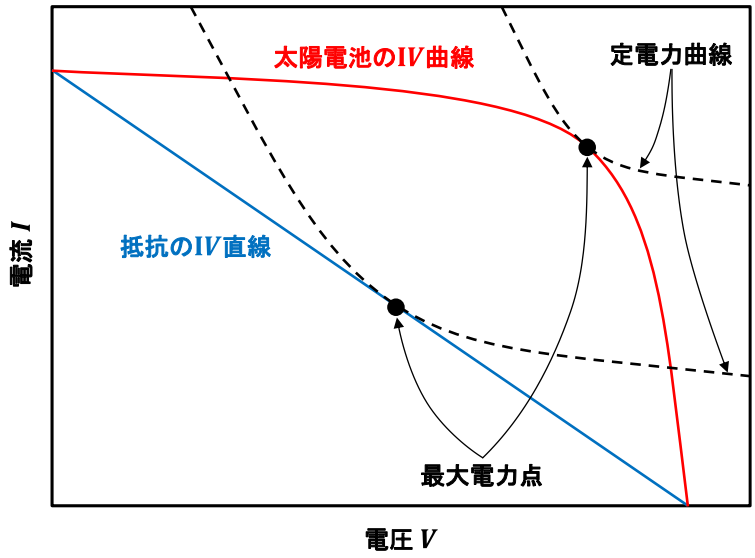


**図 3.9　太陽電池の電圧-電流特性**

以上のことを踏まえれば、太陽光パネルから鉛蓄電池へ上手に充電が行われなかったのは、定電流充電過程における制御回路の出力電圧の昇圧に対して太陽光パネルからの供給電力が不足したからであることが予想される。ここで、鉛蓄電池への出力時の出力電力とパネル電圧の関係を図 3.8 に追加し、それぞれの出力時の出力電力とパネル電圧の関係を比較するグラフを図 3.10 に示す。この図を見ると、10 Ω抵抗器を用いたときの最大出力電力は約 12 W であるが、鉛蓄電池を用いたときの最大出力電力は約 5 W であり、出力先が鉛蓄電池か 10 Ω抵抗器かによって最大発電電力が大きく異なっていることが分かる。最大発電電力にこのような差が生じるのは、発電電力は出力側の抵抗値によって変化するからだと予想した。

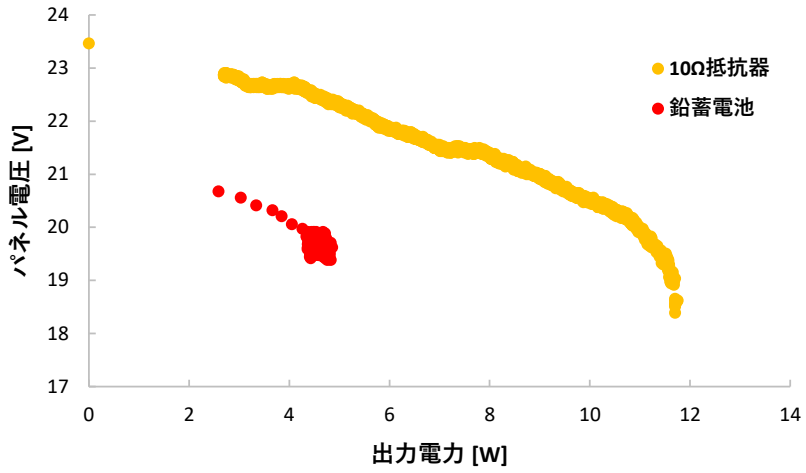


図 3.10

そこで 4 Ω抵抗器で同様に昇降圧実験を行い、図 3.10 に 4 Ω抵抗器を用いたときの出力電力とパネル電圧グラフの関係追加したグラフを図 3.11 に示す。この図を見ると、4 Ω抵抗器を用いたときの最大出力電力は約 10 W であり、出力先の抵抗値が減少することによって最大発電電力は減少することが分かる。また、鉛蓄電池の最大出力電力は 4 Ω抵抗器を用いたときよりも遥かに小さいことから、鉛蓄電池のインピーダンスは 4 Ω よりも遥かに小さいことが予想される。さらに、鉛蓄電池のインピーダンスは充電率によって変化をするため、それに伴い太陽光パネルの最大発電電力も変化することが予想される。以上のことから、正常な充電が行われなかったのは、制御プログラムが太陽光パネルの特性に対応していなかったためであり、鉛蓄電池の最大充電電力ではなく、太陽光パネルの最大発電電力に合わせた昇降圧制御プログラムを開発しなければならないことが明らかとなった。

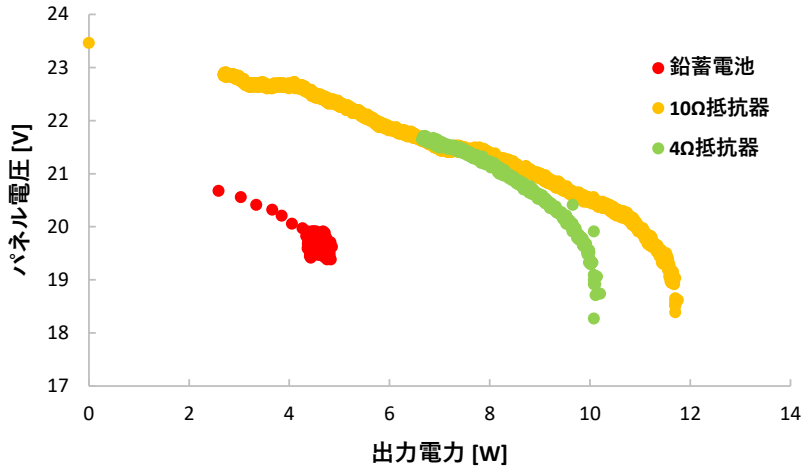


図 3.11　4 Ω と 10 Ω の抵抗器への出力電力とパネル電圧の関係

## 3.6. 太陽光発電プログラムの開発

太陽光パネルの発電電力を取得するには、太陽光パネルからの供給電流を測定する必要がある。しかしながら先に記したように、太陽光パネルから充電回路への入力電流を測定するために差動オペアンプを利用して電流値を測定する手段を検討したが上手くいかなかった。そこで、太陽光パネルからの発電電力を測定せずに出力電力の変動から発電電力を推測して制御するプログラムの構築を目指す。

3.4 節で示した太陽光パネルから鉛蓄電池への充電実験の結果と 3.5 節に行った太陽光パネルの電力出力特性の結果から、当たり前ではあるが、太陽光パネルの出力電力以上に鉛蓄電池へ電力供給することは不可能であり、太陽光パネルの最大出力電力を供給しているにもかかわらず、さらに出力電圧を昇圧しようとしても鉛蓄電池へ電力を供給することが出来ないことが分かっている。そこで、設定出力電圧と実際の出力電圧を比較して 0.3 V 以上のずれが生じた際に設定出力電圧の降圧を行う動作を追加することで、太陽光パネルの最大電力点を上回る出力電力の要求を防ぐことを目的としたプログラムを追加することとした。充電率が上がればインピーダンスは増加し、それに伴い太陽光パネルの出力電力は大きくなるが、本プログラムであれば入力側の入力電力が変化しても鉛蓄電池の定格電圧・電流値内での充電を行えるようになる。

新たに開発した充電プログラムを用いて充電実験を行った結果を図 3.12 に示す。実験は 2022/12/08 の 14 時頃に雲に太陽光が遮られることのない晴天下で行った。

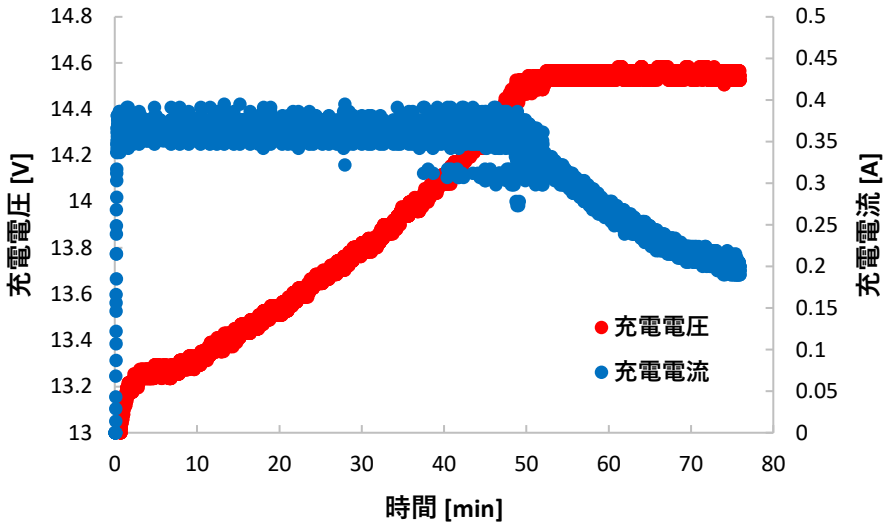


**図 3.12　太陽光発電記録 2**

図 3.12 を見ていただくと分かるが、本充電電力図は理想的な充電電力図と同じ形を描くことから、先に記したプログラムの改良によって太陽光パネルから鉛蓄電池への理想的な充電動作が行われていることが確認された。3.3 節の安定化電源を用いた鉛蓄電池への充電時と比べ、太陽光パネルでは供給電力が一定値ではないことと、出力電圧を操作する動作条件を追加したことにより昇降圧制御が滑らかになり、出力電圧・電流値が二値化しなかったと考えられる。

## 3.7. 一時的な電力不足の穴埋め

ユークリーターは上面に乗せた対象物の搬送を行うため、搬送中は上面に取り付けた太陽光パネルへの光を遮ることになる。そこで、コンデンサをパネルと昇降圧回路の間に接続することにより、一時的な発電電力不足を補うと同時に不安定な発電電力の平滑化を検討することとした。使用したコンデンサは電気二重層コンデンサ（VEC5R4505QG-H）（Vina Tech 製）であり、静電容量は 5 F、定格電圧は 5.4 V である。太陽光パネルの性能表の最大電圧は 18 V であるため、これを 4 つ直列で接続することで最大電圧 21.6 V、静電容量 1.25 F のコンデンサとして扱うことでコンデンサの定格電圧が 18 V を超えるように使用する。太陽光パネルと昇降圧回路の間に並列に接続し、3.6 節で開発した新たな動作プログラムを使用し、実際に充電実験を行った。LCD で充電の様子を見ながら、記録を開始して 1 分を過ぎたあたりで安定した充電が行われていると判断し、太陽光を遮った。充電記録を図 3.13 に示す。実験は 2022/12/22 の 12 時頃に雲に太陽光が遮られることのない晴天下で行った。

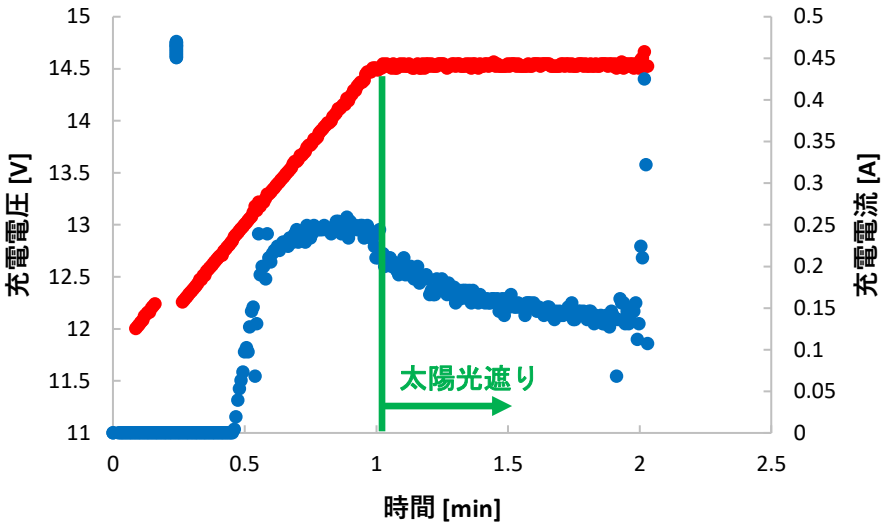


**図 3.13　コンデンサを利用した太陽光充電記録**

鉛蓄電池への充電記録を確認すると、使用した鉛蓄電池が空状態ではなかったため定電圧制御による充電が行われていることが分かる。太陽光を遮った後も定電圧充電が約 1 分間に渡り続けられていることが確認された。搬送により太陽光パネルが遮られるのは高々数秒程度と予想されるので、本節で開発した新回路を用いることで一時的な電力不足を補うことは十分可能だと考えられる。また、3.6 節の実験で得られた充電記録（図 3.12）と比較しても充電電圧・電流ともに安定した値をとり続けていることから、本新開発回路を用いることで当初の目論み通り不安定な発電電力の平滑化を実現できていることが確認された。

# 第4章　通信制御システムの改良

## 4.1. 従来の通信システム

ユークリーターは制御用 PC、動作用ユニット、その二つを繋ぐ中継器によって構成されており、PC・中継器間は一対一接続によるシリアル通信で行い、中継器・動作ユニット間は CAN バスへの複数ノード接続による CAN 通信で行う。現状動作ユニットは最大 6 台であり、動作実験時には従来の通信システムでももちろん問題は見られない。しかし、ユークリーターは基本構想では少なくとも数十〜数百台の動作ユニットの使用を想定した搬送システムであり、それほどの台数での CAN 通信による信号の伝送では、電圧降下により全てのユニットに信号を正確に送信することができない可能性が予想される。そこで、対ノイズ性に優れネットワークの再構築が容易という CAN 通信の強みを活かしたまま、数万台の同時通信を可能とする通信システムの構築を図った。

## 4.2. 改良案

従来使用していた CAN 通信用トランシーバは MCP2561 であり、ISO11898 規格でノードの最大数は 32 と規定され、ユークリーターの使用予定台数には遠く及ばない。そこで、改良案として「要所で電圧を上げ直す」「CAN バス数を増やす」の 2 つから検討をした。

1 つ目の「要所で電圧を上げ直す」という方法では、CAN トランシーバから電送された信号を電圧が落ちすぎない適度な距離離れたユニットが受信し、その受信したユニットが同じデータを送信することを繰り返すことで常に高い電圧を維持しながらデータを上流から下流へ伝送する。そのため、ノード数をどれだけ増やしたとしても電圧降下が起こる心配はない。しかし、CAN バスは閉じたループになっており、一度送った送信データが消せるわけでないので、同じデータがいくつも CAN バス内を回り続けることとなる。つまり、同じ内容のデータでも受信済みなのか未受信なのかの区別をつけなければ、電圧を上げ直すために複製されたデータが、永久的に複製され続けることになる。また、同じユニットを数と配置のみを変えることで自在に使用範囲を変更できるユークリーターの長所を生かすには、どこで電圧を上げ直すのかを仕様要望に応じて検討しなければならず、システムも複雑になってしまう。また、たとえ適度な間隔毎という制約を設けず単純に全ユニットに本機能を持たせた場合、少なくとも設置位置に関する懸念はなくなるが、制御電力が大幅に上がってしまうという問題が生じてしまう。

そこで、2 つ目に挙げた CAN バス数を増やすという方法を採用した。元々の通信システムとして、PC から中継器を通して各ノードへ CAN 通信を行っていたので、中継器から複数の CAN バスへ接続し、各 CAN バスからは 32 ノードまでを上限としてシステムを構築することとする。

通信システムのイメージを図 4.1 に示す。

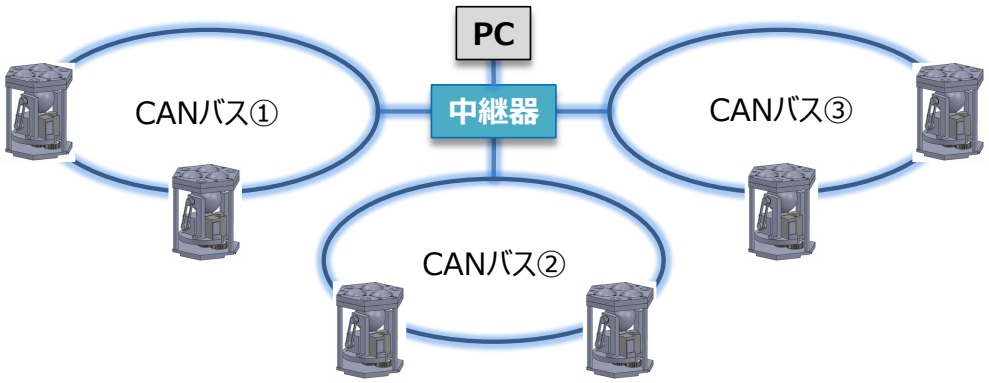


**図 4.1　通信システムイメージ**

## 4.3. 中継器作製

中継器を改良するにあたり、制御用のマイコンと CAN 通信を行うための CAN コントローラを SPI 通信によって繋ぐため、1 バス増やすのに 2 本の信号線を要する。そのため、従来使用していた Arduino NANO では接続可能なピン数が少なかったため、Arduino シリーズとは別の無線素子を内蔵するワンボードマイコン ESP32 への変更することとした。ESP32 のマイコンの出力電圧が 5V から 3.3V に変わるため、CAN トランシーバを MCP2561 から MCP2562 へ変更した。ESP32 であれば最大 8 バスの CAN バスの接続が可能である。実験的にまず 3 バスに対応する中継器を作製した。 作製した中継器を図 4.2 に、回路図を図 4.3 示す。

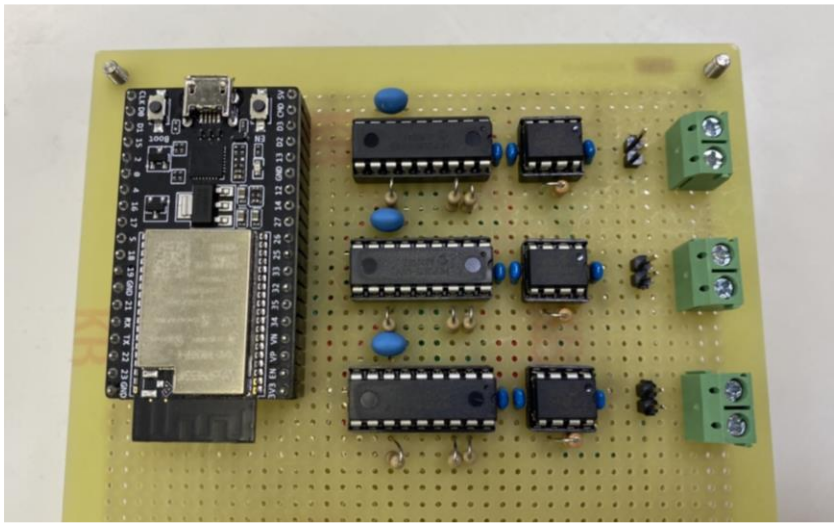

**図 4.2　3 バスに対応する中継器回路基板**

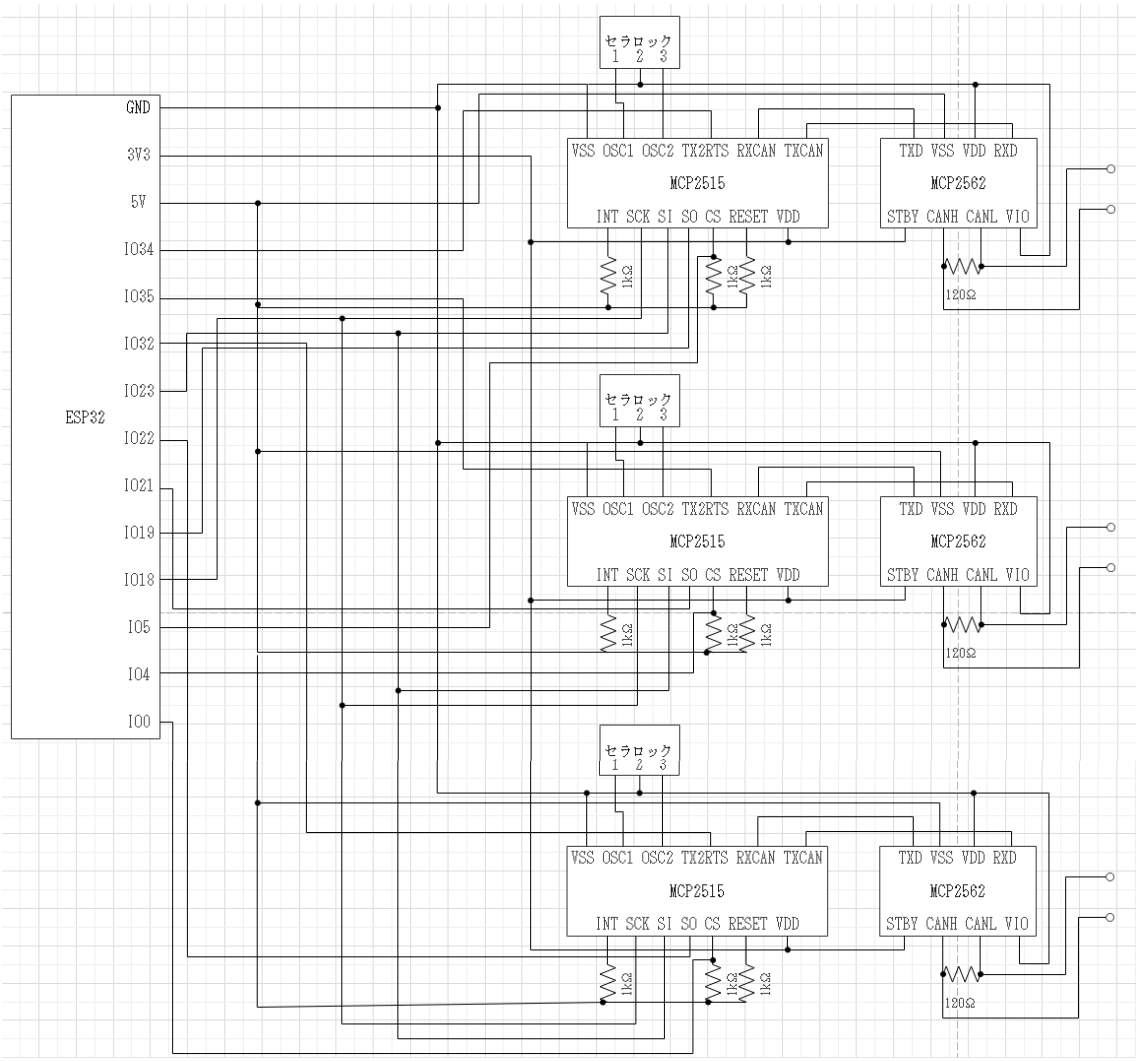


**図 4.3　中継器回路基板の回路図**

## 4.4. 動作実験

ユニットのアドレスは 2 進数で与えており、回路上では 8 つのスイッチの ON/OFF により決定している。前 4 桁が x 座標、後ろ 4 桁が y 座標に対応させた。ユニットはスイッチ係上 7 台しか用意することができなかったため、1 つの CAN バスですべてのユニットを連携させることが可能ではあったが、本実験ではせっかくなので中継器を介して各ユニットを操作することを目標とした。そこで、アスイッチ前 4 桁を CAN バスナンバーとユニットアドレスの x 座標値を紐付けて設定した。これにより、PC からユニットへの送信データは中継器において使用する CAN バスの選択を行い、必要な CAN バスにのみデータの送信を行うスイッチング機能を付与できる。ユニットのアドレスと対応 CAN バスのイメージを図 4.4 に示す。

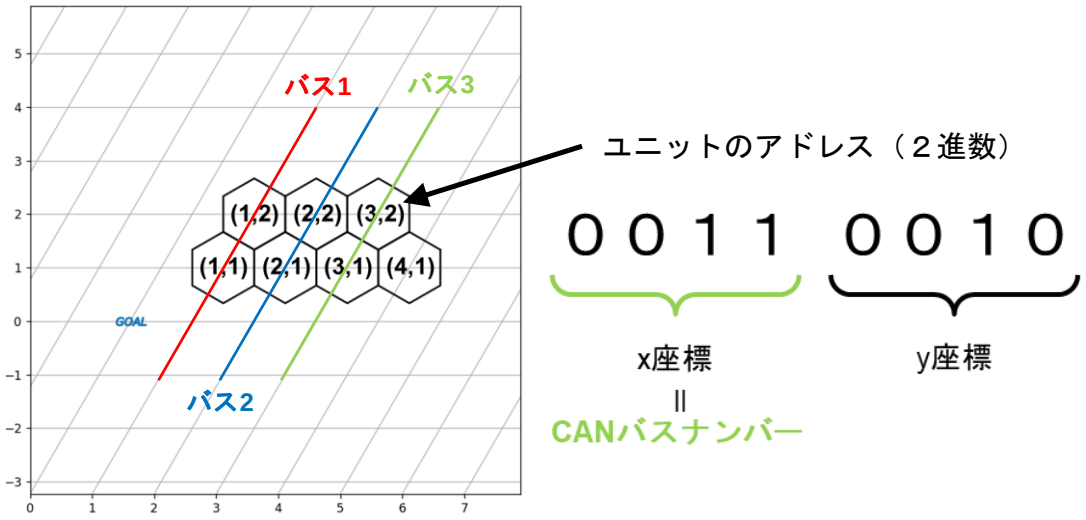


図 4.4　ユニットアドレスと対応する CAN バスのイメージ

3 つの CAN バスを利用し、1 バスにつき 2 台のユニットを接続してユークリーターの動作実験を行った。PC からユニットの動作命令を送信しユニットを動作させることで PC→ユニット間の送受信が正確に行われているかの確認と、ユニット側から定期的に文字列を送信し PC 側で表示させることで、ユニット→PC 間の送受信が行われているかの確認を行い、複数の CAN バスを利用した相互通信が行えるか検証した。結果、どちらの通信も正常に行われていることが確認され、CAN バスの増設によるユニットの接続可能台数の増加が十分に見込める通信形態を構築することに成功した。

## 4.5. データチェック

CAN バスを分散させることにより通信可能台数の増加を行ったが、PC-中継器は以前と変わらず通信経路は 1 本であるため、ユニットの接続可能台数の増加による通信データ増加に伴い、データの欠損・消失が起こる可能性を考慮し、受信時に通信データが正常に送受信を行えているか確認するためのデータチェックを行う工程を 2 つ追加することにした。1 つ目は、送信データをすべて数字化し足し合わせた時の末尾 1 桁をデータの末尾に付属させ送信を行い、受信側も同様に受信データの末尾以外を足し合わせた数の末尾 1 桁と受信データの末尾の数字を比べることで、通信データに異常が無いか確かめる方法である。2 つ目は送信データに 0～9 の番号を順に割り振って通信データに添付し、番号の順番に受信が行われているかを確認することで、通信データが消失していないかを確かめる方法である。この時、中継器では 10 個の送信データは一時的に保存をし、11 個目以降の送信データ作成時に 0 から順に保存データの上書きを行う。上記の方法で異常が見られた際には、中継器

に対して異常があった番号のデータの再送信要求を行い、通信データに抜けがないよう確実な送受信を行えるように通信プログラムを作成した。データチェック工程のため、通信データにチェック用の情報を付与したデータイメージを図 4.5 に示す。

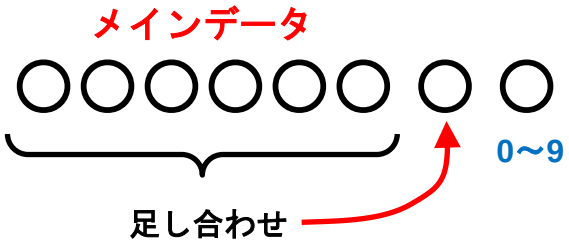


**図 4.5　データチェックを付与したシリアル通信データ**

# 第5章　自動制御システム

## 5.1. センサの搭載

機械による自動搬送を行うにあたって、搬送対象物の位置や大きさを把握することが求められる。そのためには外部情報を取得するための何らかの入力装置をシステムに組み込む必要がある。小型の装置の敷き詰めのみで搬送システムの構築を可能とするために、外部装置としての入力装置は作製せず、ユニット自身に組み込むことで搬送物の認識を可能とするシステムの開発を行った。

### 5.1.1. センサの選別

入力装置として使用する機器を、カメラ、超音波センサ、圧力センサ、赤外線センサの中から検討した。

カメラはリアルタイムに捉えた映像から画像認識によって搬送対象物の認識を行う最もスタンダードな方法であり、搬送対象物の形・大きさ・位置を正確にとらえることが出来る。しかし、上から見下ろす形でユークリーターの使用範囲全域を捉える必要があるため、搬送用ユニットとは別の入力用装置としてシステムに組み込まなければならない。さらに画像認識による高度な処理能力を必要とするなど情報処理コストも要する。

超音波センサは超音波を発し、対象物に反射した超音波を取得することで対象物との距離を測ることができる。数メートルの検知が可能なため、各ユニット毎への設置ではなく一定範囲のグループに対して 1 個といった形で設置すれば必要個数を少なくすることが出来る。しかし、距離を測る為にはユニット上面に飛び出す形で横向きに設置する必要があるほか、複数使用するため別のセンサの超音波をキャッチしないようにする工夫が必要になり、同じユニットを数と配置のみ変えて敷き詰めて使用するというユークリーターのコンセプトには適していないと思われる。

圧力センサは圧力による受圧部の変形によって生じる電気抵抗変化から圧力を計測するセンサである。搬送対象物がユニットの上に存在する時はユニットに対して荷重がかかるため、ユニット下部に取り付けることで、どのユニットが搬送に必要なのかがわかる。しかし、圧力センサを組み込むには受圧部が変形させられるほどの構造的なゆるみを持たせることが必要な上、モータ駆動の振動による誤計測を防ぐような機構の設計や選別することのできるプログラムの作成が必要になり、設計のハードルが高い。しかしながら将来的に体重移動などで進行方向を決められるようなシステムを構築するのなら、圧力センサは最も適したセンサであると思われる。

赤外線センサは検討していた候補の中で最も小型であり、システムに対してもとても単純で組み込みやすい。例えばユニット上面の搬送に使用しない隙間に搭載することが容易で

あり、上向きに設置することでユニットの上に搬送対象物があることを判定できる。搬送物認識のプロセスが最も簡易的である反面、設置するセンサ数・位置・判定処理によって正確性が左右される。

いずれにせよ本研究では赤外線センサをユークリーターに搭載することにした。もちろんほかのセンサに関してもユークリーターに導入する価値はあると思われるが、次世代の課題としておく。

使用する赤外線センサを図 5.1 に示す。本実験では反射型フォトセンサを採用した。反射型フォトセンサは赤外光 LED から発した赤外光を対象物に反射させ、帰ってきた赤外光によるフォトトランジスタの抵抗値の変化を電圧値として読み取ることで、対象物の有無を判定するセンサである。Arduino NANO でフォトトランジスタの抵抗値の変化を 0〜1023 のアナログ値で読み取る。反射光がないときが 1023 とした。また赤外線は黒色に吸収されてしまうため、黒色の物体も感知できるように閾値を 800 に設定することで感度を調整した。

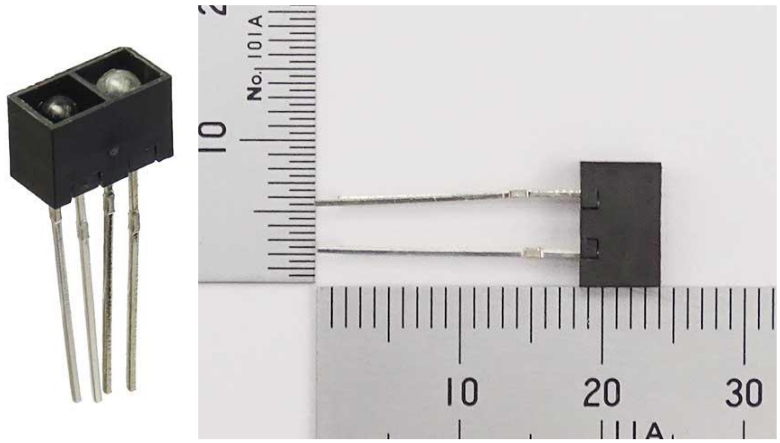


**図 5.1　フォトリフレクタ（LBR-127HLD）**

### 5.1.2. センサの組み込み構造

赤外線センサを組み込むにあたり、ユニットのどの位置に赤外線センサを組み込むのが最適かを検討した。まず、ユニットの真上に搬送対象物が乗っているという判定をするにはユニット中心部に設置するのが良いことは容易に想像できる。また、球体型ユークリーターは中心から点対象に配置された 3 つの小球体により搬送を行っているため、中心に物体がかぶらずとも搬送に携わる場合がある。そのため、センサを設置する場所が中心部だけでは不十分であり、六角形の角部にも 1 つ設置することとした。使用する際には六角形のユニットは隙間なく敷き詰められるので、角部に設置されたセンサは 2 つのユニットに隣接し、実

質的に1ユニットに対し中心部に1つと角部に3つのセンサが設置されたのと等価となる。もちろん更にもう一つの角部に1つセンサを追加すればより緻密にすることもできる。

本研究では角部へのセンサの設置に際し、2パターンの設置案の提案をした。1つは搬送用球体から遠い空白スペースに入れ込む案（図5.1）であり、この案では現状のユークリーターの構造として何もない空間であったため、3つのユニットのちょうど中心になるように設置することが容易である。2つ目は搬送用球体に近い角に入れ込む案（図5.2）である。その場合、搬送動作を行うユニットか否かの判断は本質的には搬送対象物が搬送用球体に接しているかどうかを判別することであるので、1つ目の案に比べセンサの配置が搬送用球体に近い本案の方がその判断を行うのに適しているかもしれない。しかし、構造上ユニット上面を支える柱がある場所になるので、角部のセンサを 3 つのユニットのちょうど中心になるよう設置するには構造を見直す必要がある。

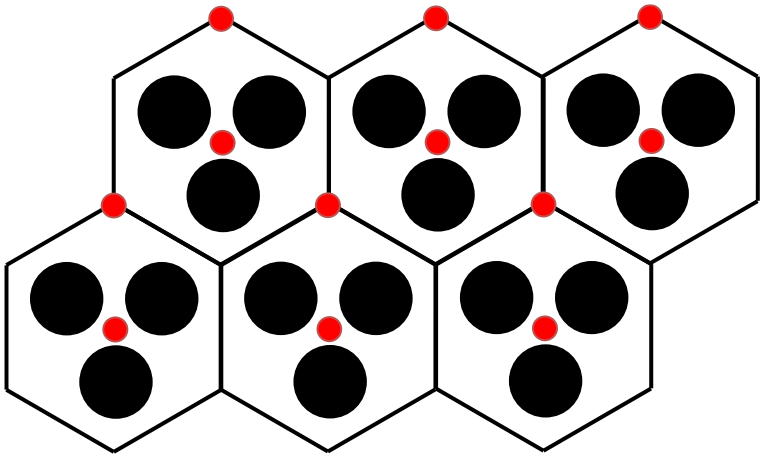

**図 5.2　センサ設置案 1**

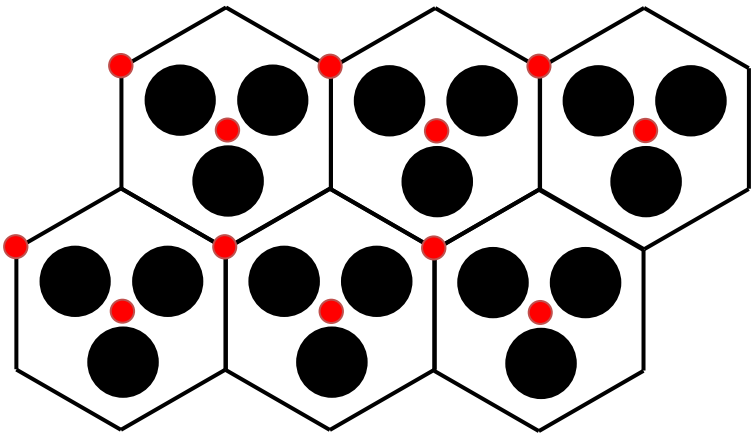

**図 5.3　センサ設置案 2**

いずれの案にしても、センサの設置を行うためにはユニットの構造改良が必要となる。そこでまず、その機構の改良が比較的簡単なセンサ配置案 1 から試験しセンサの導入の有意性を調査することとした。しかしながら、センサ配置案 1 では先に記す太陽光パネルの設置箇所とかぶる位置にセンサを組み込むことになるので、案 2 も後に検証することとした。

## 5.2. 制御プログラム

自動搬送システムの原案を作成するにあたり、搬送物認識プロセスを 1 から構築する必要がある。そこでシステム全体の情報をまとめて俯瞰し、デバックが容易となる集中制御形態を採用することとした。集中制御形態では、各ユニットが赤外線センサによって取得した搬送物の位置情報を PC に収集し、そのデータを基に搬送ルートを決定する。その後、搬送に必要なユニットを判別し、対象ユニットのみに動作命令を送信することで、必要最低限のユニット動作によって搬送対象物を目的地への搬送を可能とする自動制御を構想した。

次に位置座標値の設定であるが、先に記したユニット毎に単純に 1 ずつ変化させるアドレスを振るアドレス値と六角形ユニットの中心位置座標値を同じ値として扱うと、中心部センサは整数として扱えるが角部センサは分数（1/3 や 2/3 など）となってしまうため、位置座標値はアドレス値の 3 倍にすることですべてのセンサの位置座標値が整数値として扱えるようにした。センサ反応時または反応消失時にユニットに搭載された制御基板は自身のアドレス値とともに中心と角のどちらのセンサかの情報を送信し、PC は受け取った情報から反応しているセンサの位置座標を割り出す。どちらのセンサなのかを表す値を S と置き、中心部は S=0、角部は S= 1 とする。ユニットのアドレスを $(x, y)$ と表すと、60° 斜交座標系でのセンサの位置座標 $(x_{60°}, y_{60°})$ は $x_{60°} = 3x - S$ ， $y_{60°} = 3y + 2S$ で求められる。アドレス 1−1 のユニットにおけるセンサの位置座標を図 5.4 に示す。

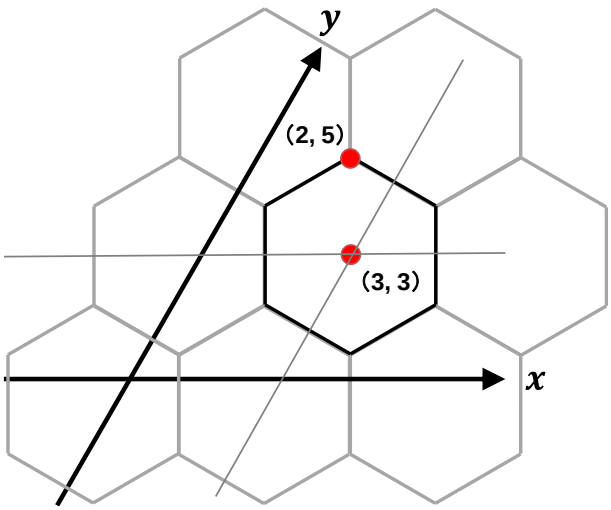


**図 5.4　ユニット 1−1 のセンサ位置座標**

搬送に必要なユニットの判別をするには搬送物の位置・大きさを認識しなければならない。そこで、反応のある全てのセンサ座標の x 座標の平均値と y 座標の平均値から中心点を求め、そこから最も遠いセンサ座標までの距離を半径とした円を搬送対象物の存在する範囲として扱うこととする。以降本編ではこの円を搬送物存在予測円と名付ける。

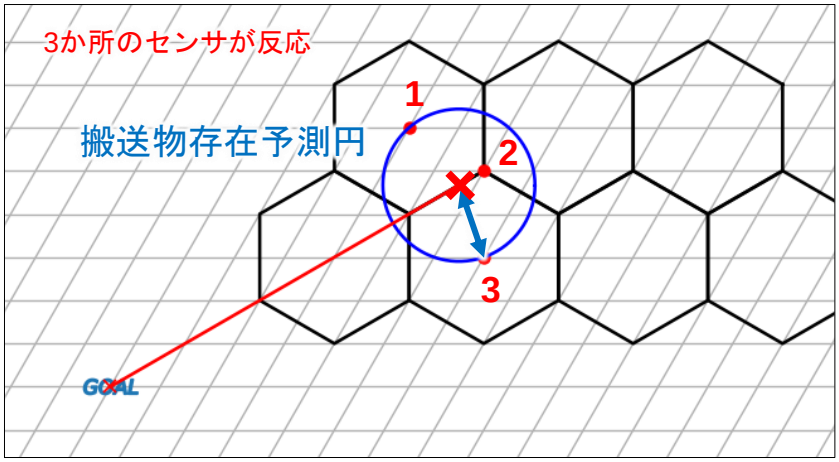


**図 5.5　搬送物認識プロセスのイメージ図**

中心点と目的地点を結ぶ直線を搬送ルートとし、直線の傾きをサーボモータの駆動角度として計算する。搬送物存在予測円の線上にあるユニットと、反応している中心センサを搭載している全てのユニットを搬送に必要なユニットとして扱う。対象のユニットアドレスにサーボモータ駆動角度と DC モータ出力を送信する。このとき、動作させるユニットのアドレスをリストに保存しておく。センサの情報が変更されるたびに上記の処理を行い、搬送ルートの再計算と動作ユニットを更新する。前回動作時に保存した動作ユニットと、更新された動作ユニットを比較し、動作に必要のなくなったユニットの選別を行い、そのユニットアドレスにはサーボモータ角度をホームポジションに戻し、DC モータ出力を 0 にする初期化動作命令を送信する。これらの処理により、リアルタイムに搬送物を追従して位置情報の認識を行い、現時点での最短搬送ルートへの修正を行いながら、必要最小数のユニットで搬送を行うプログラムを、先行研究で作成された CAN 通信用コントローラのプログラム(2.3.4)に加筆する形で作成を行った。自動搬送用に作成した PC・中継器・ユニットのプログラムのフローチャートを図 5.6 に示す。

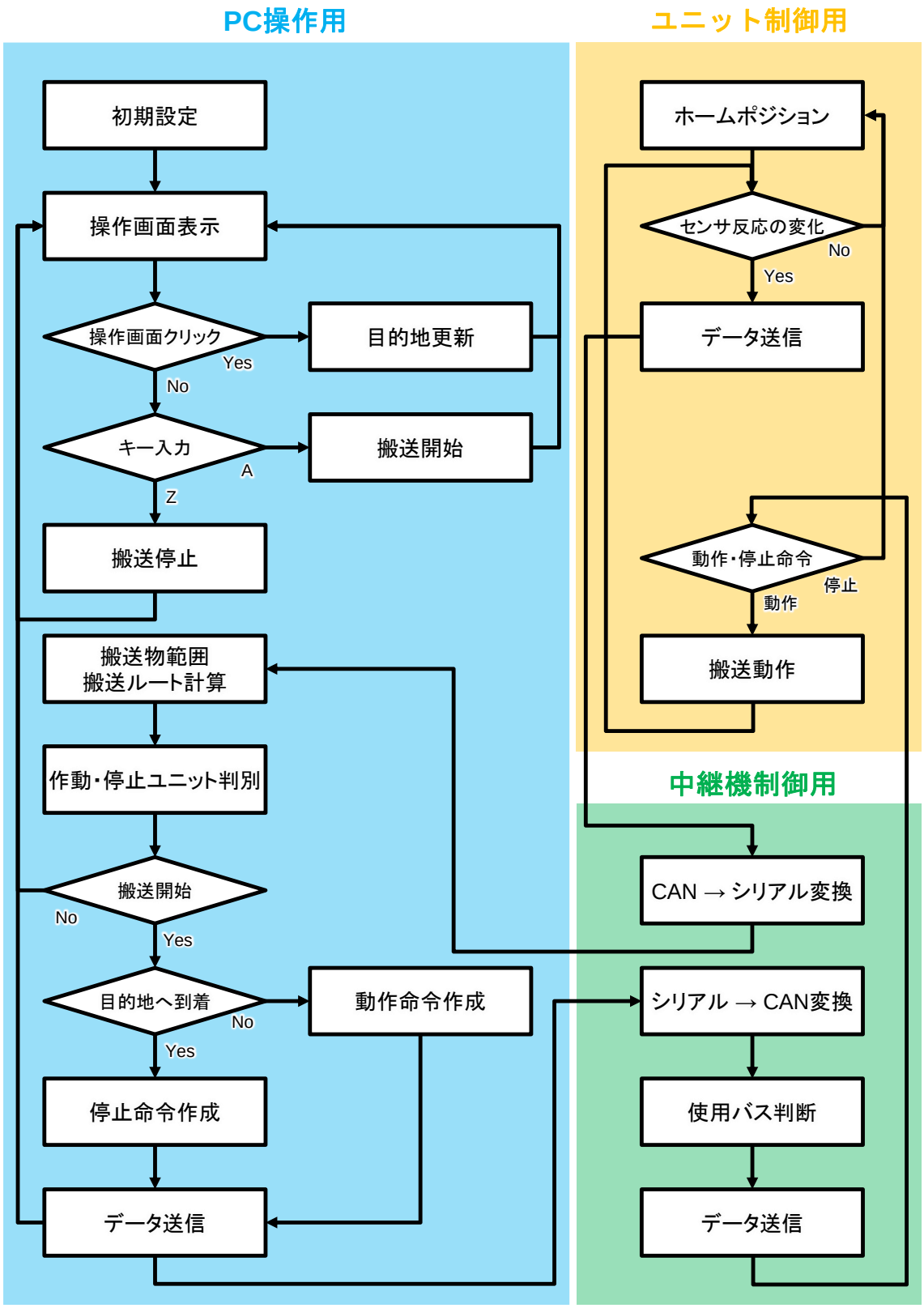


図 5.6　ユークリーター用自動搬送プログラムのフローチャート

## 5.3. 搬送実験

ユニットを 6 台使用し、DC モータの出力を 40%に制限して搬送実験を行った。DC モータの出力を制限したのは、搬送速度が速すぎると情報の処理が間に合わず搬送ルートの修正が間に合わなくなる可能性を考慮したことと、軌道の修正を行う様子を確認しやすくするための二つの理由からである。ユニットのアドレスは左下にあるユニットを 1−1、右上にあるユニットを 3−2 とした。また、搬送開始位置をユニット 3−2 として搬送対象物を乗せ、目的地をユニット 1−2 の上に設定した。ユニットの対角の距離（140 ㎜）より一回り大きい直径 150 ㎜の円形プレートを搬送物として用意した。搬送実験の様子を図 5.7 に示す。

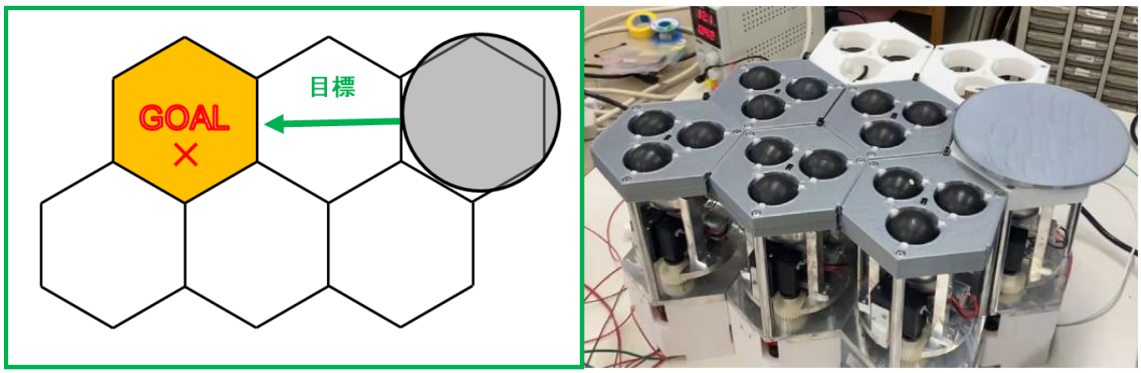


**図 5.7　搬送実験の様子**

本来ユニットを敷き詰めた際には 1 ユニットにつき 3 つの角部センサが存在するが、動作実験時は搬送対象物を初期位置に設定した際に、本来あるべき角部センサが存在せず、実際の搬送物の大きさより小さく認識を行ったため、目標搬送方向とは少しずれた方向へ搬送が開始された。搬送開始時の搬送物存在予測円と搬送ルートを図 5.8 に示す。

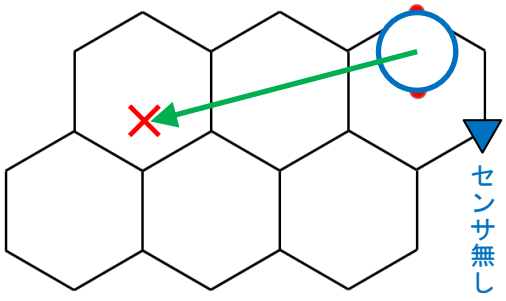


**図 5.8　搬送開始時の搬送物認識**

その後、搬送物が移動することでセンサによる位置情報が更新され、実際の搬送物の大きさと位置に近い搬送物存在予測円が表示されるようになった。それにより搬送ルート・動作ユニットの再計算が逐次進行し、都度軌道修正され目的地までしっかりと搬送対象物の搬送を行えることが確認できた。搬送実験時の搬送物の目標搬送ルートと実際に行われた搬送ルートを図 5.9 に示す。

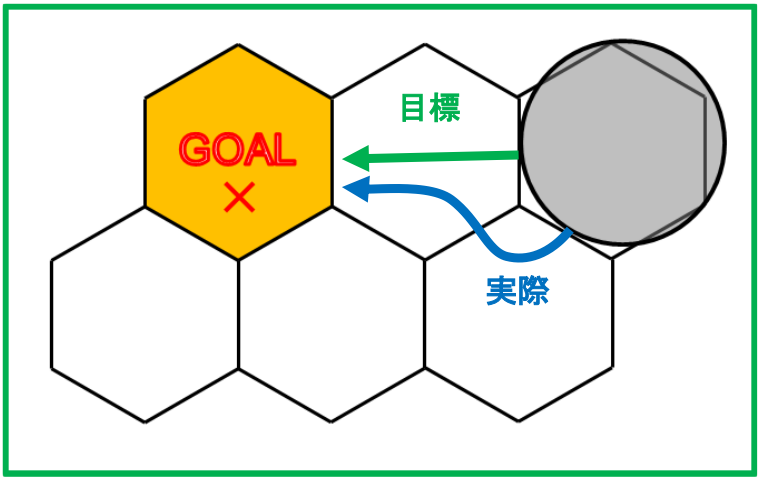


**図 5.9　搬送目標と実際のルート 1**

自動搬送実験（正常動作）の動画
https://youtu.be/m873yVgJK-M

また、搬送実験時に意図せず機器の故障が起こったことにより、ユニット 2－1 が動作命令による予定搬送方向とは全く異なる方向へ搬送するというトラブルが起こった。しかしながら搬送対象物は先の実験とは異なる軌道を通っていたが、最終的には目的地に搬送させることが確認された。搬送方向が異なっていたとしても、ユニットは自身のユニット上から外へ向けて搬送するような動作を行っているため、必ず正常に動作を行えるユニット上へ運ぶことができる。つまり、1 つ 2 つのユニットが故障していたとしても目的地への搬送が可能であることが明らかとなった。1 ユニットの故障時の搬送ルートを図 5.10 に示す。

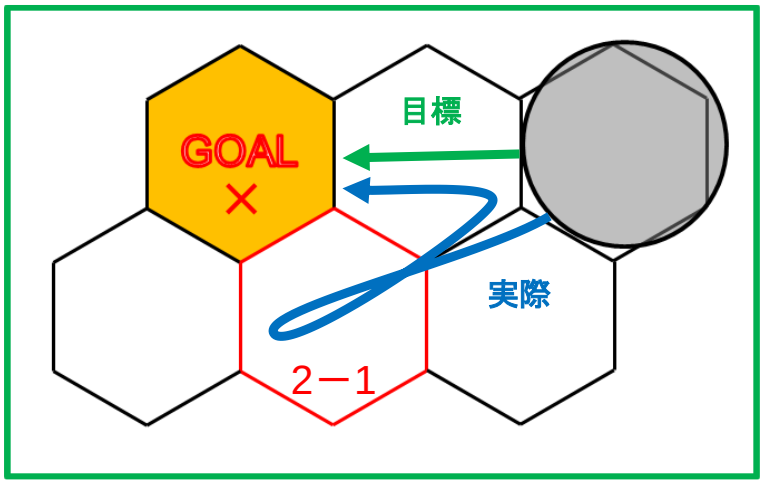


図 5.10　搬送目標と実際のルート 2

自動搬送実験（一部機能不全）の動画
https://youtu.be/AD2Pc9EgcI4

## 5.4. まとめ

以上より、本章では搬送物の認識プロセス、動作ユニットの判定、搬送ルートの計算による単体搬送物の自動搬送プログラムの開発に取り組み、それらの原案を構築することができた。もちろんまだまだ改善するべき点もあり、例えば応答速度高速化、搬送方向の修正・搬送停止動作の即応性の改善などである。実際応答の遅れにより搬送方向の修正・搬送停止動作の遅れが見られている。つまりシステムとしての搬送速度の上限が応答速さ、すなわち制御通信の速度及びそれらのデータ処理速度によって決まってしまってしまうことが判明した。もし搬送物が多くなればセンサの処理情報が増えるので応答速度は更に遅くなると考えられる。応答を早くするには、通信速度の改善が必要となる。ソフトウェアで対応することも 1 つの方法ではあるが、ハードウェアによる対応も検討してもよい。例えば、PC—中継器間のシリアル通信を LAN 通信に変更、システムプログラムを Python から C 言語に変更、中央型制御を分散型制御に変更するなどいくつかの方法が提案できる。

# 第6章　結言

本研究ではユークリーターをユニット郡として見たときに発生する課題の解決や、複数装置の連携による自律搬送システムの構築に取り組んだ。

まず、球体型ユークリーターを敷き詰めた際に発生する長方形の空きスペースに、太陽光パネルの搭載を行うことによりユークリーターへの機能の付与を図った。そのために、鉛蓄電池への充電用安定化回路の作製と充電制御プログラムの開発を行い、太陽光発電による安定した電力供給システムの構築を行った。

次に、数万台にも及ぶユニットの同時通信制御を想定し、そのような通信を可能とするシステム形態の作成を図った。従来ユークリーターに採用していた通信方法である CAN 通信をそのまま利用し、CAN バスを複数に分配することで通信可能台数の大幅な増加を行った。

最後に、複数ユークリーターを一つの装置としてとして連携させて、機械自身による制御処理によって目的地へ搬送を行う自立システムの開発に取り組み、その原型の開発を図った。１ユニットにつき赤外線センサを２個搭載し、センサで取得した情報を PC に収集することで搬送対象物の位置・大きさ計算を行う。そのデータを基に目的地までの搬送ルートと搬送に必要な動作ユニットを判別し、各ユニットに動作命令を送信することで単体搬送対象物の自動搬送を可能とした。これにより機械制御による自動搬送システムの原案を完成させた。

今後は複数搬送物に対応させ、導入の第一目標である倉庫などでの運用に向けて開発を進めてもらい、本技術を近い将来我々の日常生活に組み込むまで発展させることを期待する。

# 参考文献


[1] 藤子・F・不二雄，“映画ドラえもん　のび太と銀河超特急”，(1996)

[2] beckhoff ウェブサイト：https://www.beckhoff.com/ja-jp/products/motion/xplanar-planar-motor-system/

[3] Cellumation ウェブサイト：https://cellumation.com/products/celluveyor/

[4] 伊藤電機株式会社ウェブサイト：https://www.itohdenki.co.jp/products/mcs.html

[5] Walt Disney Imagineering ウェブサイト：https://www.designboom.com/technology/disney-holotile-floor-treadmill-vr-lanny-smoot-01-23-2024/

[6] 吉本翔斗：「未来的移動手段を想定した球体による革新的駆動伝達機構の提案」高知工科大学　卒業論文　2016.03

[7] 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」高知工科大学大学院 修士論文，2017.03

[8] 秋山将太郎：「球体を用いた伝達機構の高効率化」高知工科大学大学院 学士論文，2017

[9] 佐藤雅也：「次世代移動手段「ベアリングロード」の構造解析　高知工科大学　卒業論文　2017.03

[10] 竹中克昭：「次世代型全方向輸送機構の開発」高知工科大学大学院　修士論文　2019.03

[11] 長嶋晋也：「球体駆動機構を用いた次世代移動手段の開発」高知工科大学　卒業論文 2019.03

[12] 狩野大輝：「球体を用いた敷詰型全方向搬送機構の開発」 高知工科大学大学院 修士論文 2020.03

[13] 鈴鹿紅音：「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」　高知工科大学大学院　修士論文 2020.03

[14] 亀岡正樹：「全方向搬送装置へのセンサの設置と制御機構開発」　高知工科大学　卒業論文　2021.03

[15] 高井友輝：「全方向搬送機器における太陽電池を用いた電力の自給自足化」　高知工科大学　卒業論文　2021.03

[16] 石井和磨：「ベルトを用いた全方向搬送装置の開発」　高知工科大学大学院　修士論文 2021.03

[17] 澤田陸斗：「全方向運搬装置のルート決定アルゴリズムの開発に向けた基礎研究」　高知工科大学　卒業論文　2022.03

[18] 和仁原季也：「全方向搬送装置ユークリーターの制御システムの開発」　高知工科大学大学院　修士論文　2022.09

[19] 高井友輝：「ベルトを用いた全方向搬送装置における張力調整機構の開発」　高知工科大学大学院　修士論文　2023 .03

[20] The Matplotlib development team. SkewT-logP diagram: using transforms and custom projections. https://matplotlib.org/2.2.3/gallery/api/skewt.html, (参照 2022/08/16).

# 付録

## 太陽光発電から充電プログラム

コードは Arduino で記載。

太陽パネルから受け取った電力を鉛蓄電池へ 0.02 V ずつ電圧を昇降圧し、出力が 14.4 V、0.36 A を超えないように調整。設定出力電圧と実際の出力電圧に大きな差が見られたら降圧を行う。

```
/*
  D0   serial_TX
  D1   serial_RX
  D2
  D3
  D4
  D5
  D6
  D7
  D8
  D9
  D10 SPI_CS(SD カード)
  D11 SPI_MOSI(SD カード)
  D12 SPI_MISO(SD カード)
  D13 SPI_CLK(SD カード)
  A0   半固定抵抗(10kΩ)
  A1   入力電圧(0.0～30.5[V])   1.0kΩ と 5.1kΩ で分圧     5.0*(1.0+5.1)=30.5
  A2   出力電圧(0.0～20.0[V])   1.0kΩ と 3.0kΩ で分圧     5.0*(1.0+3.0)=20.0
  A3   出力電流(0.1Ω の両端電圧) オペアンプで 10.1 倍
  A4   I2C_SDA(LCD & DCDC コンバータ)
  A5   I2C_SCL(LCD & DCDC コンバータ)
  A6
  A7   入力電流(0.1Ω の両端電圧) オペアンプで 10 倍
  */
#include   <Wire.h>                          // I2C 通信で LCD と DCDC コンバータ
#include   <SPI.h>                           // SPI 通信で SD カード
#include   <SD.h>                            // SD カード
#include <LiquidCrystal_I2C.h>               // LCD を使用するため
LiquidCrystal_I2C lcd(0x3F,16,2);            // I2C アドレスは 0x3F
#define       dccAD 0x62                     // DC-DC コンバータ SC8721 の I2C アドレス
const int cs = 10;                           // SDC Arduino UNO NANO では 10
int           b0, b1, b2, v;                 // v は DCDC コンバータの出力電圧　(5.0+0.02*b0)[V]
int           vi0, vi1, vi2, vi3;            //入力電圧
int           vo0, vo1, vo2, vo3;            //出力電圧
int           ii0, ii1, ii2, ii3;            //入力電流
int           io0, io1, io2, io3;            //出力電流
float         p,p0;
char m1[32], m2[32];
unsigned long time;
// I2C デバイスへの書き込み関数
void writeDT(byte ad,byte dt)
{
     Wire.beginTransmission(0x62);
     Wire.write(ad);
     Wire.write(dt);
     Wire.endTransmission();
     delay(10);
}
void setup(void)
{
     Serial.begin(9600);
     Wire.begin();                           // I2C の初期化
     SD.begin();
     b0=350;                                 // SC8721 出力電圧変数の初期化　出力電圧　5.0+0.02*b0=5.0[V]
     lcd.init();
     lcd.backlight();
     lcd.setCursor(0,0);
     lcd.print("solar-charge7");
     //SD 出力設定
     File dataFile = SD.open("charging.csv", FILE_WRITE);
     dataFile.println("solar-charge7");
     dataFile.println("time[ms],Iin[mA],Vin[mV],Iout[mA],Vout[mV],P[W],b0");
     dataFile.close();
     p0=0;
}
```

```
// この loop()内で b0 の値を変更すれば DCDC コンバータの出力電圧が変化します。
// なお、DCDC コンバータの入力は 5[V]～15[V]位の AC アダプタを使用する。
void loop(void)
{
    byte   bm,bl;
    // 16bit の a0 を上位 8bit(bm)と下位 2bit に分割
    bm=(byte)(b0 >> 2);                                  //  上位 8bit
    bl=(byte)(b0 & 0x0003) | 0x18;                       //  下位 2bit+0x18
    // SC8721 レジスタへの書き込み
    writeDT(0x01,0xf0);                                  // CSO
    writeDT(0x02,0x00);                                  // SLOPE_COMP
    writeDT(0x03,bm);                                    // VOUT_SET_MSB
    writeDT(0x04,bl);                                    // VOUT_SET_LSB
    writeDT(0x05,0x02);                                  // GLOBAL_CTRL
    writeDT(0x06,0x80);                                  // SYS_SET
    // ===0x04 レジスタの設定===
    //      5-7bit   リザーブ
    //      4bit     ボリューム調整          デジタル調整は'1'をセット
    //      3bit     VOUT-SET のマスタビット   変更可能は'1'をセット
    //      2bit     出力電圧方向ビット         5[V]以上は'0'をセット
    //      0-1bit   出力電圧設定レジスタの下位 2 ビット

    //dcdc コンバータへの入力電圧
    vi0=analogRead(A1);
    //実際の入力の 1/6 を読み取り(0~5V を 0~1023) 計算は(×6×5) 単位は mV
    vi1=float(vi0*30.0*1000.0/1023.0);
    vi2=vi1/1000;
    vi3=(vi1-vi2*1000)/10;
    //dcdc コンバータへの入力電流（0.1Ω にかかる電圧をオペアンプで 10 倍）を読み取る
    ii0=analogRead(A7);
    //0.1Ω にかかる電圧を 10 倍で読み取り(0~5V を 0~1023)  計算は(×5÷10÷0.1) 単位は mV=mA
    ii1=float(ii0*5.0*1000.0/1023.0);
    ii2=ii1/1000;
    ii3=(ii1-ii2*1000)/10;
    //dcdc コンバータからの出力電圧を読み取る
    vo0=analogRead(A2);
    //実際の入力の 1/4 を読み取り(0~5V を 0~1023) 計算は(×4×5) 単位は mV
    vo1=float(vo0*20.0*1000.0/1023.0);
    vo2=vo1/1000;
    vo3=(vo1-vo2*1000)/10;
    //負荷にかかる電圧(オペアンプで 10 倍)を読み取る
    io0=analogRead(A3);
    //0.1Ω にかかる電圧を 10 倍で読み取り(0~5V を 0~1023)  計算は(×5÷10÷0.1) 単位は mV=mA
    io1=float(io0*5.0*1000.0/1023.0);
    io2=io1/1000;
    io3=(io1-io2*1000)/10;
    //パネル電力を計算
    p0=ii1*vi1;
    //プログラム開始からの経過時間を読み取る
    time = millis();
    //LCD に表示する
    //左上   入力電流[mA]                  右上   入力電圧[mv]
    sprintf(m1,"      %1d.%02dA,%02d.%02dV",ii2,ii3,vi2,vi3);
    lcd.setCursor(0,0);
    lcd.print(m1);
    //左下   出力電流[mA]                  右下   出力電圧[mV]
    sprintf(m2,"%3d,%1d.%02dA,%02d.%02dV",b0,io2,io3,vo2,vo3);
    lcd.setCursor(0,1);
    lcd.print(m2);
/*バッテリーの充電条件に合わせたもの
    //電圧の設定条件
    if(vo1 < 14400) b0=b0+1;
    else b0=b0-1;
    if(io1 > 360) b0=b0-2;
    //太陽光パネルの最大効率化に合わせたもの
    p=(ii1/1000.0)*(vi1/1000.0);                         //測定値からパネル電力を算出
    if(p >= p0) b0=b0+1;                                 //前の電力より大きくなっていれば電圧を上げる
    else b0=b0-1;
    p0=p;
*/
    //パネルの供給電力を算出、10W 以上を保つ
    p=(ii1/1000.0)*(vi1/1000.0);                         //測定値からパネル電力を算出
    if(p <= 10) b0=b0+1;                                 //パネル電力が 10W より小さければ電圧を上げる
    else b0=b0-1;
    //バッテリーの充電電流は 0.36A を超えないように（大事をとって 0.35 を超えるかで判断）
    if(io1 > 350) b0=b0-1;
    //バッテリーン電圧が 15V を超えないように（14.5 を超えるかで判断）
    if(vo1 > 14500) b0=b0-1;
    //設定した理想出力電圧と測定した出力電圧を比較して差が大きすぎたら設定電圧を下げる
    v=5000+20*b0;
    if(v-vo1 > 300) b0=b0-1;
    //充電完了,プログラムは続行
```

```
if(vo1 > 13500 && io1 < 10)
{
  /* LCD に表示 */
  Serial.println(io1);
  lcd.setCursor(0,0);
  lcd.print("charging comp");
  delay(1000);                                    //充電完了後は（とりあえず）1 秒間隔でプログラム回す
  /* SD カードに書き込み */
  File dataFile = SD.open("charging.csv", FILE_WRITE);
  dataFile.println("charging complete");
  dataFile.println("Vin,Iout,Vout");
  dataFile.close();
  /* フロート電圧をかけ続ける */
  for(;;)
  {
    if(io1 > 0) b0=b0-1;
    bm=(byte)(b0 >> 2);                           //  上位 8bit
    bl=(byte)(b0 & 0x0003) | 0x18;                //  下位 2bit+0x18
    /* SC8721 レジスタへの書き込み */
    writeDT(0x01,0xf0);                           // CSO
    writeDT(0x02,0x00);                           // SLOPE_COMP
    writeDT(0x03,bm);                             // VOUT_SET_MSB
    writeDT(0x04,bl);                             // VOUT_SET_LSB
    writeDT(0x05,0x02);                           // GLOBAL_CTRL
    writeDT(0x06,0x80);                           // SYS_SET
    //dcdc コンバータへの入力電圧
    vi0=analogRead(A1);
    //実際の入力の 1/6 を読み取り(0~5V を 0~1023）計算は(×6×5）単位は mV
    vi1=float(vi0*30.0*1000.0/1023.0);
    vi2=vi1/1000;
    vi3=(vi1-vi2*1000)/10;
    //dcdc コンバータへの入力電流（0.1Ω にかかる電圧をオペアンプで 10 倍）を読み取る
    ii0=analogRead(A7);
    //0.1Ω にかかる電圧を 10 倍で読み取り(0~5V を 0~1023)　計算は(×5÷10÷0.1）単位は mV=mA
    ii1=float(ii0*5.0*1000.0/1023.0);
    ii2=ii1/1000;
    ii3=(ii1-ii2*1000)/10;
    //dcdc コンバータからの出力電圧を読み取る
    vo0=analogRead(A2);
    //実際の入力の 1/4 を読み取り(0~5V を 0~1023）計算は(×4×5）単位は mV
    vo1=float(vo0*20.0*1000.0/1023.0);
    vo2=vo1/1000;
    vo3=(vo1-vo2*1000)/10;
    //負荷にかかる電圧(オペアンプで 10 倍)を読み取る
    io0=analogRead(A3);
    //0.1Ω にかかる電圧を 10 倍で読み取り(0~5V を 0~1023)　計算は(×5÷10÷0.1）単位は mV=mA
    io1=float(io0*5.0*1000.0/1023.0);
    io2=io1/1000;
    io3=(io1-io2*1000)/10;
    //パネル電力を計算
    p0=ii1*vi1;
    //プログラム開始からの経過時間を読み取る
    time = millis();
    //LCD に表示する
    //左上　入力電流[mA]　　右上　入力電圧[mv]
    sprintf(m1,"      %1d.%02dA,%02d.%02dV",ii2,ii3,vi2,vi3);
    lcd.setCursor(0,0);
    lcd.print(m1);
    //左下　出力電流[mA]　　右下　出力電圧[mV]
    sprintf(m2,"%3d,%1d.%02dA,%02d.%02dV",b0,io2,io3,vo2,vo3);
    lcd.setCursor(0,1);
    lcd.print(m2);
    /* SD カードに書き込み */
    File dataFile = SD.open("charging.csv", FILE_WRITE);
    if (dataFile)
    {
      dataFile.print(time);                       //時間（プログラム開始からの経過時間）
      dataFile.print(",");
      dataFile.print(ii1);                        //入力電流（パネルの出力電流）
      dataFile.print(",");
      dataFile.print(vi1);                        //入力電圧（パネルの出力電圧）
      dataFile.print(",");
      dataFile.print(io1);                        //（バッテリーへの）出力電流
      dataFile.print(",");
      dataFile.print(vo1);                        //（バッテリーへの）出力電圧
      dataFile.print(",");
      dataFile.print(p0);                         //パネル電力
      dataFile.print(",");
      dataFile.println(b0);                       //コンバータに送る b0 の値
    }
    dataFile.close();
  }
```

```
    }
    /* SD カードに書き込み */
    File dataFile = SD.open("charging.csv", FILE_WRITE);
    if (dataFile)
    {
      dataFile.print(time);                    //時間（プログラム開始からの経過時間）
      dataFile.print(",");
      dataFile.print(ii1);                     //入力電流（パネルの出力電流）
      dataFile.print(",");
      dataFile.print(vi1);                     //入力電圧（パネルの出力電圧）
      dataFile.print(",");
      dataFile.print(io1);                     //（バッテリーへの）出力電流
      dataFile.print(",");
      dataFile.print(vo1);                     //（バッテリーへの）出力電圧
      dataFile.print(",");
      dataFile.print(p0);                      //パネル電力
      dataFile.print(",");
      dataFile.println(b0);                    //コンバータに送る b0 の値
    }
    dataFile.close();
    delay(250);
}
```

PC による制御プログラム

コードは Python で記載。ユニットのアドレス情報は Excel ファイルから参照。

PC 画面上に操作画面を表示、マウスクリックにて目的地の入力を行う。シリアル通信により送られてきたデータを異常がないかチェックし、異常があれば再送信要求を送り返す。異常がなければセンサ情報を取得後、搬送物の位置・大きさの算出をし、操作画面に図を表示する。それと同時に搬送ルートと動作ユニットを割り出して対象のユニットに対して動作命令を送信する。

```
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.path as mtplpath
import serial
import keyboard
import sys
from my_module import my_skewt
#データチェック用ナンバー
num = 10
#センサのアドレス値を格納する配列
sens_list=[]
motion_unit=[]
motion_unit_list=[]
stop_unit_list=[]
last_unit_list=[]
cx=[]
cy=[]
class Unit:
    """ユニットの個別パラメータを格納しておくクラス.
    Attributes
    ----------
    id : str
        ユニットの ID => 8bit の 2 進数
    position_x : int
        ユニットの原点の x 座標
    position_y : int
        ユニットの原点の y 座標
    angle : int
        ユニットの駆動角度
    velocity : int
        ユニットの駆動速度
    profile_x : list of float
        ユニット外形の各頂点の x 座標
    profile_y : list of float
        ユニット外形の各頂点の y 座標
    coutour : list
        内外判定用の頂点座標
    """
    def __init__(self, id: str, position_x: int, position_y: int):
        """initialize.
        Parameters
        ----------
            id : str
                ユニットの ID => 8bit の 2 進数
            position_x : int
                ユニットの x 座標
            position_y : int
                ユニットの y 座標
        """
        self.id = id
        self.position_x = position_x
        self.position_y = position_y
        self.velocity = 0
        self.angle = 90
        # 斜交座標系での六角形座標への補正量
        hex_profile_x = np.array([1, -1, -2, -1, 1, 2, 1])
        hex_profile_y = np.array([1, 2, 1, -1, -2, -1, 1])
        # 中心点の調整
        self.profile_x = hex_profile_x + float(position_x)
        self.profile_y = hex_profile_y + float(position_y)
        # 内外判定用
        self.contour = []
        for i in range(7):
```

```
                self.contour.append([self.profile_x[i], self.profile_y[i]])
        def draw(self) -> None:
            """外形をグラフに描画する関数"""
            plt.plot(self.profile_x, self.profile_y, color='black')
            return None
        def set_on_route(self, velocity: int, angle: int) -> None:
            """内外判定で True の時に速度、角度に変更があれば値を更新し、変更フラグを立てる関数"""
            if self.velocity != velocity:
                self.velocity = velocity
            if self.angle != angle:
                self.angle = angle
            return None
        def set_off(self) -> None:
            """ユニットをオフにして原点回帰する関数"""
            self.velocity = 0
            self.angle = 90
            return None
    class FlexArray:
        """アレイ化したユニット群のパラメータを格納しておくクラス.
        Attributes
        ----------
        unit : list
            アレイ内のユニットクラスを継承
        See Also
        --------
        Unit : class
            ユニットの個別パラメータを格納しておくクラス
        """
        def __init__(self, array_config):
            """initialize.
            Parameters
            ----------
            array_config : dataflame
                array_config_2.csv から取得したデータフレーム
            """
            self.unit = []
            for i in range(len(array_config)):
                self.unit.append(Unit(array_config.id[i], array_config.position_x[i], array_config.position_y[i]))
    def skewt_plt_setup() -> tuple:
        """my_skewt モジュールを用いた 60 度斜交座標系のセットアップを行う関数.
        Returns
        -------
        fig : Figure
        ax : Axes
        """
        fig = plt.figure(figsize=(7, 7))
        ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], projection='skewx')
        ax.set_aspect(np.sqrt(3)/2, 'datalim')  # 'equal'では高さが合わない -> √3/2 倍する
        ax.grid(True)
        ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
        ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
        return fig, ax
    def oblique2cartesian(oblique_x: float, oblique_y: float) -> list:
        """斜交座標系から直交座標系への座標変換を行う関数.
        変換行列
            x' = x cos θ + y cos ϕ
            y' = x sin θ + y sin ϕ
        ただし θ : X'軸と X 軸のなす角度 = 0°, ϕ : X'軸と Y 軸のなす角度 = 60°
        Parameters
        ----------
        oblique_x : float
            斜交座標系での x 座標
        oblique_y : float
            斜交座標系での y 座標
        Returns
        -------
        [cartesian_x, cartesian_y] : list of float
            cartesian_x : float
                直交座標系での x 座標
            cartesian_y : float
                直交座標系での y 座標
        See Also
        --------
        cartesian2oblique : 直交座標系から斜交座標系への座標変換を行う関数.
        """
        cartesian_x = oblique_x + 1/2*oblique_y
        cartesian_y = np.sqrt(3)/2*oblique_y
        return [cartesian_x, cartesian_y]
    def cartesian2oblique(cartesian_x: float, cartesian_y: float) -> list:
        """直交座標系から斜交座標系への座標変換を行う関数.
        変換行列
            x = 1/det(T)*(x' sin ϕ - y' cos ϕ )
```

```
        y = 1/det(T)*(-x' sin θ + y' cos θ )
    ただし det(T)=cos θ sin ϕ -cos ϕ sin θ , θ = 0° , ϕ = 60°
    Parameters
    ----------
    cartesian_x : float
        直交座標系での x 座標
    cartesian_y : float
        直交座標系での y 座標
    Returns
    -------
    [oblique_x, oblique_y] : list of float
        oblique_x : float
            斜交座標系での x 座標
        oblique_y : float
            斜交座標系での y 座標
    See Also
    --------
    oblique2cartesian : 斜交座標系から直交座標系への座標変換を行う関数.
    """
    oblique_x = cartesian_x - 1/np.sqrt(3)*cartesian_y
    oblique_y = 2/np.sqrt(3)*cartesian_y
    return [oblique_x, oblique_y]
def get_angle_vector(current_position: list, destination_position: list) -> float:
    """現在地から目的地への偏角を取得する関数.
    現在地から目的地へのベクトルを複素数に変換し、複素平面上で偏角を取得する
    直交座標系での角度を求めるときは引数に渡す前に座標変換しておくこと
    Parameters
    ----------
    current_position : list of float
        現在地の位置ベクトル[x 座標, y 座標]
    destination_position : list of float
        目的地の位置ベクトル[x 座標, y 座標]
    Returns
    -------
    angle : float
        現在地から目的地へのベクトルの偏角(-pi < angle < pi) [deg]
    """
    vector_complex = complex(
        destination_position[0] - current_position[0],
        destination_position[1] - current_position[1]
    )
    angle = np.angle(vector_complex, deg=True)
    # サーボの正回転方向とグラフ上の正回転方向を合わせるための補正
    if angle > 0:
        angle = 180 - angle
    else:
        angle = -180 - angle
    return angle
def get_3digits_str(value: float) -> str:
    """引数を三桁に整えて str 型で返す関数.
    Parameter
    ---------
    value : int or float
        ユニットの駆動速度または角度
    Return
    ------
    value_3digits_str : str
        0 を追加して右づめにした str 型の value
    """
    value_rounded = str(round(value))
    if len(value_rounded) == 3:
        value_3digits_str = str(value_rounded)
    elif len(value_rounded) == 2:
        value_3digits_str = "0" + str(value_rounded)
    elif len(value_rounded) == 1:
        value_3digits_str = "00" + str(value_rounded)
    else:
        return None
    return value_3digits_str
def get_message(unit_id: str, velocity: float, angle: float) -> bytes:
    """ジャンクション基板へ送信するメッセージを作成する関数.
    送信先の ID(8Byte), 駆動方向(1Byte), 駆動速度(3Byte), 駆動角度(3Byte), 終端文字(1Byte)の ASCII センテンス
を作成する
    Parameter
    ---------
    unit_id : str
        送信先ユニットの ID
    velocity : int or float
        ユニットの駆動速度
    angle : float
        ユニットの駆動角度
    Return
```

```
        ------
        message : bytes
            ジャンクション基板へ送信する 16Byte の ASCII センテンス
            "XXXXYYYYDVVVAAA;"
                XXXXYYYY : ID
                D : 駆動方向(1 : 正転, 0 : 逆転)
                VVV : 駆動速度
                AAA : 駆動角度
                ; : 終端文字
        """
        if angle < 0:
            angle = angle + 180
            direction = 0
        else:
            direction = 1
        message_str = str(unit_id) + str(direction) + str(get_3digits_str(velocity)) + str(get_3digits_str(angle)) + str(";")
        message = message_str.encode("ascii")
        return message
def onclick(event) -> None:
        """グラフ上をクリックした時に実行する関数.
        Notes
        -----
        クリック位置を目的地に設定し、グラフの描画を更新する
        """
        global goal_unit, sens_mean
        click_position = np.array([event.xdata, event.ydata])
        if (click_position[0] != None) and (click_position[1] != None):
            destination_point[0] = click_position[0]
            destination_point[1] = click_position[1]
            [destination_point_cartesian[0],  destination_point_cartesian[1]]  =  oblique2cartesian(destination_point[0],
destination_point[1])
            goal.set_data(destination_point[0], destination_point[1])
            center_point.set_data([sens_mean[0], destination_point[0]], [sens_mean[1], destination_point[1]])
        fig.canvas.draw_idle()
        #荷物の中心がどのユニットの上にあるか判断
        for unit_i in array.unit:
                #中心点のユニット内外判定
                if mtplpath.Path(unit_i.contour).contains_point([event.xdata, event.ydata]) == True:
                    #荷物中心があるユニット id を取得
                    goal_unit=format(unit_i.id.zfill(8))
        print("目的地のユニット", goal_unit)
        return None
def motion(xdata: int, ydata: int, sens: int, snum: int) -> None:
        """
        センサ情報を受信したときの処理
        """
        print("def motion start")
        #ユニットのアドレス
        id_x = xdata
        id_y = ydata
        #センサの座標
        xdata = xdata*3 - 2*snum
        ydata = ydata*3 + 1*snum
        #mouse_position = np.array([xdata, ydata])
        shape_recognition(xdata, ydata, sens)
        #id の設定、4 桁 2 進数に変換　アレイデータからとってくるべきか？
        id_x2=format(id_x, '04b')
        id_y2=format(id_y, '04b')
        sens_id=str(id_x2) + str(id_y2)

        fig.canvas.draw_idle()
        print("def motion finish")
        return None
def move_unit():
        #キー入力で'a'が押されていて、センサ反応が 2 個以上なら搬送開始
        global motion_unit_list, stop_unit_list, last_unit_list, sens_list
        if len(sens_list) >= 2:
            mouse_position_cartesian = np.array(oblique2cartesian(sens_mean[0], sens_mean[1]))
            angle = get_angle_vector(mouse_position_cartesian, destination_point_cartesian)
            #搬送速度 0~255
            velocity = 50
            unit_i.set_on_route(round(velocity), round(angle))
            #円プロットが内判定のユニット全てに動作命令を送る
            for unit_id in motion_unit_list:
                message = get_message(unit_id, unit_i.velocity, unit_i.angle)
                #time.sleep(0.1)
                ser.write(message)
                print("送信メッセージ", message)  # debug
            monved = True
        #搬送停止中、もしくはセンサ反応が 2 個未満なら動作停止命令
        else:
            unit_i.set_off()
```

```
            #円プロットが内判定のユニット全てに動作命令を送る
            if last_unit_list != None:
                for unit_id in last_unit_list:
                    message = get_message(unit_id, unit_i.velocity, unit_i.angle)
                    #time.sleep(0.1)
                    ser.write(message)
                    print("送信メッセージ", message)  # debug
        # 搬送に必要なくなるユニットを停止させる
        if stop_unit_list != []:
            unit_i.set_off()
            #円プロットが外判定になったユニット全てに停止命令を送る
            for unit_id in stop_unit_list:
                message = get_message(unit_id, unit_i.velocity, unit_i.angle)
                #time.sleep(0.1)
                ser.write(message)
                print("送信メッセージ", message)  # debug

        return None
def stop_all():
        global last_unit_list, motion_unit_list
        unit_i.set_off()
        #ユニット全てに動作命令を送る
        if last_unit_list != None:
            for unit_id in last_unit_list:
                message = get_message(unit_id, unit_i.velocity, unit_i.angle)
                #time.sleep(0.1)
                ser.write(message)
                print("送信メッセージ", message)  # debug

        if motion_unit_list != None:
            for unit_id in motion_unit_list:
                message = get_message(unit_id, unit_i.velocity, unit_i.angle)
                #time.sleep(0.1)
                ser.write(message)
                print("送信メッセージ", message)  # debug
def shape_recognition(xdata: int, ydata: int, sens: int)-> list:
        """
        センサ情報から荷物のだいたいの大きさを割り出し、グラフに円を描く
        x_data : int
            新しく取得したセンサの x 座標(60 度斜交座標)
        y_data : int
            新しく取得したセンサの y 座標(60 度斜交座標)
        sens_list : list
            センサの座標(60 度斜交座標)
        x : int
            一時的な計算用の変数
        y : int
            一時的な計算用の変数
        cx : list
            グラフに描く円のプロットする 1 点の x 座標の配列(直交座標)
        cy : list
            グラフに描く円のプロットする 1 点の y 座標の配列(直交座標)
        sens_mean : list
            中心点の座標(60 度斜交座標)
        x_max : int
            中心から最も遠いセンサの x 方向の距離(斜交のまま、問題あり？)
        y_max : int
            中心から最も遠いセンサの y 方向の距離(斜交のまま、問題あり？)
        r : float
            荷物の範囲を円に見立てたときの半径
        cartesian_x : float
            グラフに描く円のプロットする 1 点の x 座標(直交座標)
        cartesian_y : float
            グラフに描く円のプロットする 1 点の y 座標(直交座標)
        oblique_x : float
            グラフに描く円のプロットする 1 点の x 座標(60 度斜交座標)
        oblique_y : float
            グラフに描く円のプロットする 1 点の x 座標(60 度斜交座標)
        """
        global sens_list, sens_mean, motion_unit_list, stop_unit_list
        global cx, cy, center_unit
        if sens == 49:
            #センサ ON
            sens_list.append([xdata, ydata])
        else:
            #センサ OFF
            sens_list.remove([xdata, ydata])
        #反応センサ位置プロット用の x,y 座標の配列（x,y で別々に）作成
        x = [r[0] for r in sens_list]
        y = [r[1] for r in sens_list]
        #反応センサ位置をプロット
        sens_point.set_data(x, y)
```

```
print("反応のあるセンサリスト", sens_list)

#前回の円の情報をリセット
cx.clear()
cy.clear()
#動いているユニット id を保存
last_unit_list=motion_unit_list
#ユニット情報の初期化
stop_unit_list=[]
motion_unit=[]
#反応センサが 2 個以上のとき、搬送物があると認識
if len(sens_list) >= 2:
    #行ごとの平均値を求める ＝ 中心点の座標（斜交座標）
    sens_mean=np.mean(sens_list, axis=0)
    print("中心点座標",sens_mean)
    x_max=0
    y_max=0
    #荷物の中心がどのユニットの上にあるか判断
    for unit_i in array.unit:
        #中心点のユニット内外判定
        if mtplpath.Path(unit_i.contour).contains_point(sens_mean) == True:
            #荷物中心があるユニット id を取得
            center_unit=format(unit_i.id.zfill(8))
            print("荷物中心ユニット", center_unit)

    for i in range(len(sens_list)):
        #斜交座標系を直交座標系に直し、中心点から一番遠いセンサ座標の距離を半径として求める
        x=abs((sens_list[i][0] + 1/2*sens_list[i][1]) - (sens_mean[0] + 1/2*sens_mean[1]))
        y=abs(np.sqrt(3)/2*sens_list[i][1] - np.sqrt(3)/2*sens_mean[1])
        if x+y > x_max+y_max:
            x_max=x
            y_max=y
    r = np.sqrt(x_max*x_max + y_max*y_max)
    print("搬送物半径",r)

    #中心点から求めた半径で円を描く（直交座標）
    for i in range(360):
        #斜交座標系を直交座標系に直し、直交座標上で円を計算
        cartesian_x=sens_mean[0] + 1/2*sens_mean[1] + r*np.cos(np.radians(i))
        cartesian_y=np.sqrt(3)/2*sens_mean[1] + r*np.sin(np.radians(i))
        #直交座標系を斜交座標形に直し、斜交座標系上で円を描く
        oblique_x=cartesian_x - 1/2*(2/np.sqrt(3)*cartesian_y)
        oblique_y=2/np.sqrt(3)*cartesian_y
        #円のプロット点を x と y でそれぞれ配列に保存
        cx.append(oblique_x)
        cy.append(oblique_y)
        for unit_i in array.unit:
            #円プロット点のユニット内外判定
            if mtplpath.Path(unit_i.contour).contains_point([oblique_x, oblique_y]) == True:
                #円のプロットがあるユニット id をリスト化
                motion_unit.append(format(unit_i.id.zfill(8)))
    #反応しているセンサのあるユニットは動かす
    for i in range(len(sens_list)):
        for unit_i in array.unit:
            #反応センサ点のユニット内外判定
            if mtplpath.Path(unit_i.contour).contains_point([sens_list[i][0], sens_list[i][1]]) == True:
                #反応センサがあるユニット id をリストに追加
                motion_unit.append(format(unit_i.id.zfill(8)))
    #重複しているユニット id を削除
    motion_unit_list=list(set(motion_unit))
    print("動作ユニット id=", motion_unit_list)
    print("動作中ユニット id=", last_unit_list)
    #停止させるユニット id を保存
    if last_unit_list != []:
        for i in range(len(last_unit_list)):
            result=last_unit_list[i] in motion_unit_list
            if result == False:
                stop_unit_list.append(last_unit_list[i])
    #デバック
    print("停止ユニット id=", stop_unit_list)
    #円と中心点をプロット点を更新
    circle.set_data(cx, cy)
    center_point.set_data([sens_mean[0], destination_point[0]], [sens_mean[1], destination_point[1]])
else:
    #円と中心点のプロットを削除する
    circle.set_data(cx, cy)
    center_point.set_data([],[])
    stop_unit_list=motion_unit_list
    motion_unit_list=[]
    print("荷物無し")
#グラフ上に図形を描画
fig.canvas.draw_idle()
```

```
    #動作させるユニット id のリストを返す
    #return motion_unit_list
def checknum(dnum: bytes) -> bool:
    """受信データが抜けなく番号順に受け取っているか確認する関数"""
    #dnum(文字)を dnum(数値)に変換
    dnum = dnum - 0x30
    print("date number=",dnum)
    global num

    if num == 10:
        num = dnum-1

    #確認用ナンバーを設定（9 の次は 0 になる）
    if num == 9:
        num = -1

    #番号が順番通りになっているか確認
    if dnum-1 == num:
        num = dnum
        return True
    else :
        return False
def checkSUM(cs: int, cs1: bytes, cs2: bytes) -> bool:
    """受信データが間違っていないかメインデータの総和で確かめる（チェックサム）関数"""
    #cs の下 2 桁だけ抜き出し
    cs = cs%100
    print("データ SUM",cs)
    # 2 文字の数字を 2 桁の数字に変換
    cs12 = (cs1-0x30)*10 + (cs2-0x30)
    print("チェック用 SUM",cs12)
    #チェックサム
    if cs12 == cs:
        return True
    else:
        return False

def resend(dnum: bytes, c: int):
    global num
    i=dnum-0x30-num
    #データナンバー・チェックサムともに False
    if c==0:
        for n in range(num+1, i):
            #再送信要請メッセージ作成（0000：中継器アドレス（仮)、n：データ番号）
            message=(str("0000")+str(n)+str(";")).encode("ascii")
            print("Num SUM F send",message)
            ser.write(message)
    #チェックサムが False
    if c==1:
        #データナンバーはあっているので、再度同じナンバーのデータを送信要請
        #再送信要請メッセージ作成（0000：中継器アドレス（仮)、n：データ番号）
        message=(str("0000")+str(dnum)+str(";")).encode("ascii")
        print("check SUM F send",message)
        ser.write(message)
    #データナンバーが False
    if c==2:
        for n in range(num+1, i):
            #再送信要請メッセージ作成（0000：中継器アドレス（仮)、n：データ番号）
            message=(str("0000")+str(n)+str(";")).encode("ascii")
            print("date number F send",message)
            ser.write(message)
    return
if __name__ == '__main__':
    # シリアルポートのセットアップ
    #ser = serial.Serial("/dev/cu.usbserial-110", "9600", timeout=1) # MacOS での記述方法
    ser = serial.Serial("COM2", "9600", timeout=1)                  # WindowsOS での記述方法
    time.sleep(1.0)

    # 目的地の設定
    destination_point = np.zeros(2)
    destination_point_cartesian = np.array(oblique2cartesian(destination_point[0], destination_point[1]))
    # アレイデータの取得
    array_config_data = pd.read_csv("array_config_2.csv", header=0, names=("id", "position_x", "position_y"),
dtype={"id": str})
    array = FlexArray(array_config_data)
    # グラフのセットアップ、ユニットの描画
    fig, ax = skewt_plt_setup()
    for unit_i in array.unit:
        unit_i.draw()

    # 現在地(マウスカーソルの位置)と目的地の描画
    goal, = plt.plot(destination_point[0], destination_point[1], marker="$GOAL$", markersize=25)
    goal_unit = 00000000
```

```
sens_point, =plt.plot([], [], ".", markersize=10, color="red")
circle, =plt.plot([], [], color="blue")
center_point, = plt.plot([], [], marker="x", color="red")
#  グラフ上でマウスを操作した時の処理
fig.canvas.mpl_connect("button_press_event", onclick)
#fig.canvas.mpl_connect("motion_notify_event", motion)
#  グラフ表示領域を修正
x_min, x_max = ax.get_xlim()
plt.xlim(0, x_max+3)
#最初はユニットは動かさない状態
move=False                    #動かすか否か
moved=False                   #今動いているか
updata=False                  #情報が更新されたか
center_unit = None
while True:
    #  グラフの表示
    plt.pause(0.01)

    #  ヘッダを受け取った時にシリアル受信
    if ser.read() == b'{':
        sens = ser.read(9)
        print("受信データ", sens)
        #終端文字を確認し「}」=125  だったら処理を開始
        if sens[8] == 125:
            #byte 型 3 桁のアドレス情報を 10 進数の数値に変換
            print("OK")
            #チェック用のカウント
            c=0
            #データ番号のチェック　合っていたら"True",間違っていたら"False"が返ってくる
            if checknum(sens[7]):
                print("Number True")
                c=c+1
            else:
                print("Number False")
            #チェックサム　合っていたら"Ture",間違っていたら"False"が返ってくる
            cs = sens[0] + sens[1] + sens[2] + sens[3] + sens[4]
            if checkSUM(cs,sens[5],sens[6]):
                print("SUM True")
                c=c+2
            else:
                print("SUM False")
            if c == 3:
                sens_id = (sens[0] - 0x30)*100 + (sens[1] - 0x30)*10 + (sens[2] - 0x30)
                print("ユニット id(未加工)", sens_id)
                #2 進数に変換し、x,y 座標に分離後、10 進数に戻す
                sens_x = sens_id >> 4
                sens_y = sens_id & 0b00001111
                print("ユニット id(x, y)", sens_x, sens_y)
                print("センサの ON(49)/OFF(48)", int(sens[3]))
                #def motion()  に放り込む
                motion(sens_x, sens_y, sens[3],sens[4]-0x30)
                updata = True
            else:
                resend(sens[7], c)
    #搬送開始
    if move == True:
        if updata == True:
            #搬送完了
            if goal_unit == center_unit:
                stop_all()
                move = False
                moved = False
                print("搬送完了")
            else:
                move_unit()
                updata = False
                moved = True

    #搬送停止
    else:
        if moved == True:
            stop_all()
            moved = False
            print("搬送停止")
    #搬送開始キー
    if keyboard.is_pressed('a'):
        move = True
        print("搬送開始キー押下")

    #搬送停止キー
    if keyboard.is_pressed('z'):
        move = False
```

```
        print("搬送停止キー押下")

#プログラム終了
if keyboard.is_pressed('escape'):
        sys.exit()
```

中継器用プログラム

コードは Arduino で記載。

ユニット宛のシリアル通信データを受信したら CAN 通信データ形式へ変換し、宛先の CAN バスへ CAN 通信で送信する。CAN 通信データを受信したらシリアル通信データ形式へ変換し、シリアル通信で送信する。変換したシリアルデータは 10 個まで一時保存。PC からデータの再送信要求が来たら指定されたデータを再送信する。

```
/*  ピン情報
 * 36(IO22):I2C_SCL
 * 5(IO34):SPI_INT1
 * 6(IO35):SPI_INT2
 * 7(IO32):SPI_INT3
 * 25(IO0):SPI_CS3
 * 26(IO4):SPI_CS2
 * 29(IO5):SPI_CS1
 * 30(IO18):SPI_SCK
 * 31(IO19):SPI_MISO
 * 33(IO21):I2C_SDA
 * 37(IO23):SPI_MOSI
 */
#include <stdio.h>
#include <mcp_can.h>
#include <SPI.h>
#define   CAN1_INT 34                                   // Set CAN1_INT to pin   34
#define   CAN2_INT 35                                   // Set CAN2_INT to pin   35
#define   CAN3_INT 32                                   // Set CAN3_INT to pin   32
MCP_CAN CAN1(5);                                        //29 ピン、CAN バス 1 の CS1
MCP_CAN CAN2(4);                                        //25 ピン、CAN バス 2 の CS2
MCP_CAN CAN3(0);                                        //26 ピン、CAN バス 3 の CS3
char m[128],sm[10];                                     //  文字列格納用
char ssm[10][10];
int i = 0;                                              //  文字数のカウンタ
int d=0;                                                //データ番号
int sens;                                               //受信センサ値
int snum;                                               //受信センサ番号（1：角部センサ、2：中心部センサ）
int bus, dc, sv, dr;
long unsigned int id;
unsigned char rid;                                      // rid  受信 id
unsigned char len = 0;
unsigned char buf[8];
void setup()
{
  Serial.begin(9600);
  //3 ノードの CAN 通信が正常か確認----------------------------------
  if(CAN1.begin(MCP_ANY, CAN_125KBPS, MCP_20MHZ) == CAN_OK)
  {Serial.println("1,MCP2515 Initialized Successfully!");}
  else
  {Serial.println("1,Error Initializing MCP2515...");}

  if(CAN2.begin(MCP_ANY, CAN_125KBPS, MCP_20MHZ) == CAN_OK)
  {Serial.println("2,MCP2515 Initialized Successfully!");}
  else
  {Serial.println("2,Error Initializing MCP2515...");}
  if(CAN3.begin(MCP_ANY, CAN_125KBPS, MCP_20MHZ) == CAN_OK)
  {Serial.println("3,MCP2515 Initialized Successfully!");}
  else
  {Serial.println("3,Error Initializing MCP2515...");}
  //-----------------------------------------------------------

  CAN1.setMode(MCP_NORMAL);
  CAN2.setMode(MCP_NORMAL);
  CAN3.setMode(MCP_NORMAL);
  //センサ情報の先頭文字と終端文字
  sm[0]=0x7B;
  sm[9]=0x7D;
}
//byte buf[8] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};
void loop()
{
      int che;
      unsigned char idd;
      String input;
      bool TF;
      //  データ受信したとき
      if(Serial.available())
```

```
{
    input = Serial.readStringUntil(';');

    bus = (input.charAt(0) - 0x30)*8
        + (input.charAt(1) - 0x30)*4
        + (input.charAt(2) - 0x30)*2
        + (input.charAt(3) - 0x30)*1;
    //中継器宛の受信データ
    if(bus == 0000)
    {
      TF = resend(input.charAt(4)-0x30);
    }
    //ユニット宛の受信データ
    else
    {
      id = (input.charAt(0) - 0x30)*128
        + (input.charAt(1) - 0x30)*64
        + (input.charAt(2) - 0x30)*32
        + (input.charAt(3) - 0x30)*16
        + (input.charAt(4) - 0x30)*8
        + (input.charAt(5) - 0x30)*4
        + (input.charAt(6) - 0x30)*2
        + (input.charAt(7) - 0x30)*1;
      dr = input.charAt(8)-0x30;
      dc = (input.charAt(9)-0x30)*100 + (input.charAt(10)-0x30)*10 + (input.charAt(11)-0x30)*1;
      sv = (input.charAt(12)-0x30)*100 + (input.charAt(13)-0x30)*10 + (input.charAt(14)-0x30)*1;
      TF = can_send();
    }
}
//can バスから受信した時（バス 1）
//Serial.println(digitalRead(0));   //INT(割り込み用？)の確認
//if (!digitalRead(CAN3_INT))
{
  CAN1.readMsgBuf(&id, &len, buf);
  //メッセージがリモート要求フレームであるかどうかを判断
  if ((id & 0x40000000) == 0x40000000)
  {
    //sprintf(m, " REMOTE REQUEST FRAME");
    //Serial.println(m);
  }
  else
  {
    if ((unsigned char)(id & 0x00ff) == 1)
    {
      rid=buf[0];
      sens=buf[4];
      snum=buf[5];
      //rid(char 型,3 桁,0~255,1B)を sm[1~3](char 型,3 桁,0~9,3B)に変換
      sm[1]=rid/100+0x30;
      sm[2]=rid/10%10+0x30;
      sm[3]=rid%10+0x30;

      if(sens == 1)
      {
        //Serial.print("sens ON!  アドレス:");
        sm[4]=0x31;
      }
      else
      {
        //Serial.print("sens OFF  アドレス:");
        sm[4]=0x30;
      }
      //どこのセンサか（角部 or 中心部）
      sm[5]=snum+0x30;

      //チェック SUM を生成
      che=sm[1]+sm[2]+sm[3]+sm[4]+sm[5];
      //Serial.println(che);
      sm[6]=che/10%10+0x30;
      sm[7]=che%10+0x30;

      sm[8]=d+0x30;
      //Serial.println(sm);
      //再送用に送信データを保存
      for(i=0;i<=9;i++)
      {
        ssm[d][i]=sm[i];
        Serial.println(ssm[d][i]);
      }
      Serial.write(sm);
      d++;
      id=0;
```

```
    }
  }
}
//can バスから受信した時（バス 2）
//if (!digitalRead(CAN3_INT))
{
  CAN2.readMsgBuf(&id, &len, buf);
  //メッセージがリモート要求フレームであるかどうかを判断
  if ((id & 0x40000000) == 0x40000000)
  {
    //sprintf(m, " REMOTE REQUEST FRAME");
    //Serial.println(m);
  }
  else
  {
    if ((unsigned char)(id & 0x00ff) == 1)
    {
      rid=buf[0];
      sens=buf[4];
      snum=buf[5];
      //rid(char 型,3 桁,0~255,1B)を sm[1~3](char 型,3 桁,0~9,3B)に変換
      sm[1]=rid/100+0x30;
      sm[2]=rid/10%10+0x30;
      sm[3]=rid%10+0x30;

      if(sens == 1)
      {
        //Serial.print("sens ON!  アドレス:");
        sm[4]=0x31;
      }
      else
      {
        //Serial.print("sens OFF  アドレス:");
        sm[4]=0x30;
      }
      //どこのセンサか（角部 or 中心部）
      sm[5]=snum+0x30;

      //チェック SUM を生成
      che=sm[1]+sm[2]+sm[3]+sm[4]+sm[5];
      //Serial.println(che);
      sm[6]=che/10%10+0x30;
      sm[7]=che%10+0x30;

      sm[8]=d+0x30;
      //Serial.println(sm);
      //再送用に送信データを保存
      for(i=0;i<=9;i++)
      {
        ssm[d][i]=sm[i];
        Serial.println(ssm[d][i]);
      }
      Serial.write(sm);
      d++;
      id=0;
    }
  }
}

//can バスから受信した時（バス 3）
//if (!digitalRead(CAN3_INT))
{
  CAN3.readMsgBuf(&id, &len, buf);
  //メッセージがリモート要求フレームであるかどうかを判断
  if ((id & 0x40000000) == 0x40000000)
  {
    //sprintf(m, " REMOTE REQUEST FRAME");
    //Serial.println(m);
  }
  else
  {
    if ((unsigned char)(id & 0x00ff) == 1)
    {
      rid=buf[0];
      sens=buf[4];
      snum=buf[5];
      //rid(char 型,3 桁,0~255,1B)を sm[1~3](char 型,3 桁,0~9,3B)に変換
      sm[1]=rid/100+0x30;
      sm[2]=rid/10%10+0x30;
      sm[3]=rid%10+0x30;

      if(sens == 1)
```

```
                {
                  //Serial.print("sens ON!  アドレス:");
                  sm[4]=0x31;
                }
                else
                {
                  //Serial.print("sens OFF  アドレス:");
                  sm[4]=0x30;
                }
                //どこのセンサか（角部 or 中心部）
                sm[5]=snum+0x30;
                //チェック SUM を生成
                che=sm[1]+sm[2]+sm[3]+sm[4]+sm[5];
                //Serial.println(che);
                sm[6]=che/10%10+0x30;
                sm[7]=che%10+0x30;

                sm[8]=d+0x30;
                //Serial.println(sm);
                //再送用に送信データを保存
                for(i=0;i<=9;i++)
                {
                  ssm[d][i]=sm[i];
                  Serial.println(ssm[d][i]);
                }
                Serial.write(sm);
                d++;
                id=0;
              }
            }
          }
          if(d==10)d=0;
}
//ユニットに CAN 通信で送信する関数
int can_send()
{
  unsigned char idd;
  idd=(unsigned char)id;
  // dc と sv を char 型に合わせて値を半分にする
  dc = dc/2;
  sv = sv/2;
  // sprintf(buf, "%d %d %d", idd, dc, sv);
  // Serial.println(buf);
  // if(input.charAt(8) == 0x31) dc = -dc;
  buf[0]=0;                                          //  送信元 ID=0
  buf[1]=(unsigned char)dc;                          // DC ﾓｰﾀの値
  // buf[1]=dc;                                      // DC ﾓｰﾀの値
  buf[2]=(unsigned char)sv;                          // Servo ﾓｰﾀの値
  // buf[2]=sv;                                      // Servo ﾓｰﾀの値
  // buf[3]=0;                                       // ﾘｻﾞｰﾌﾞ　未使用
  if(dr == 1)
  {
      buf[3] = (unsigned char)1;
  }
  else
  {
      buf[3] = (unsigned char)0;
  }
  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

  //アドレス値によって通信するノードを選択
  //(送信先 id，フレーム状態（0 が標準），データの長さ[Byte]，送信データ)
  if(bus == 1){CAN1.sendMsgBuf(idd, 0, 8, buf);}
  if(bus == 2){CAN2.sendMsgBuf(idd, 0, 8, buf);}
  if(bus == 3){CAN3.sendMsgBuf(idd, 0, 8, buf);}
  return true;
}
//pcに再送信する関数
int resend(int num)
{
  for(i=0;i<=9;i++)
  {
    sm[i]=ssm[num][i];
  }
  Serial.write(sm);
  return true;
}
```

ユニット用プログラム

コードは Arduino で記載。

センサに反応があれば中継器宛にセンサ情報を送信する。中継器から受信があればアドレスが自身のものと一致するかの確認を行い、一致すれば受信データ通りにモータの駆動を行う。間違っていれば受信データは捨てる。

```
/*
 * 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）   ﾌｫﾄｾﾝｻの値を入力
 * 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;
int                 rsens1, rsens2;
bool s1, s2, sens1, sens2;
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(5,OUTPUT);                                  // INA2
    pinMode(6,OUTPUT);                                  // servo
    pinMode(4,INPUT);                                   // dipSW0
    pinMode(7,INPUT);                                   // dipSW1
    pinMode(8,INPUT);                                   // dipSW2
    pinMode(9,INPUT);                                   // dipSW3
    pinMode(15,INPUT);                                  // sens 入力

    pinMode(16,INPUT);                                  // dipSW4
    pinMode(17,INPUT);                                  // dipSW5
    pinMode(18,INPUT);                                  // dipSW6
    pinMode(19,INPUT);                                  // dipSW7
    svm0.attach(6);
    analogWrite(3,0);                                   //  停止
    digitalWrite(5,0);                                  //  正転
    svm0.write(90);                                     // ﾎｰﾑﾎﾟｼﾞｼｮﾝ
    delay(1000);
    Serial.begin(9600);
    if (CAN0.begin(MCP_ANY, CAN_125KBPS, 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(9);        s0=~s0 & 0x01;
```

```
    s1=digitalRead(8);      s1=~s1 & 0x01;
    s2=digitalRead(7);      s2=~s2 & 0x01;
    s3=digitalRead(4);      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;
    //アドレス（x2 桁,y2 桁）→100x+y で 3~4 桁に変換
    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);
    s1=false;                                       //センサーのカウント用
    s2=false;                                       //センサーのカウント用
    //初期設定
}
void loop()
{
    int         dc,sv, direction;
    int         an0;
    unsigned int an;
    rsens1=analogRead(A0);
    rsens2=analogRead(A1);
    if(rsens1 > 800)sens1=false;                    //センサの閾値設定
    else sens1=true;
    if(rsens2 > 800)sens2=false;                    //センサの閾値設定
    else sens2=true;
    //Serial.println(rsens);
    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);
        //メッセージがリモート要求フレームであるかどうかを判断
        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)
            {
                //  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
                // direction = buf[3];
                // for(;;){
                svm0.write(sv);
                delay(100);

                if (buf[3] == 1)                    //  正転
                {
                    analogWrite(3,dc);
                    digitalWrite(5,0);
                }
                else if(buf[3] == 0)                //  反転
                {
                    digitalWrite(3,0);
                    analogWrite(5,dc);
                }
                // if(Serial.available() > 0)
                // {break;}
                // }
            }
        }
    }
    //  フォトセンサ 1 の反応時に CAN 通信で中継器に送信
    if (sens1 == true)
```

```
{
  if(s1 == false)
  {
    Serial.println("sencer1 ON!");

    //送信データの作成
    buf[0]=sid;                                   // 送信元 ID、ドライバーによって異なる
    buf[1]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[2]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[3]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[4]=1;                                     // センサ値
    buf[5]=1;                                     // 角部センサ
    buf[6]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[7]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    //(送信先 id, フレーム状態（0 が標準）, データの長さ[Byte], 送信データ)
    CAN0.sendMsgBuf(1, 0, 8, buf);
    s1=true;
  }
  //Serial.print(s);
}
else
{
  if(s1 == true)
  {
    Serial.print("sencer1 OFF アドレス:");
    Serial.println(sid);

    //送信データの作成
    buf[0]=sid;                                   // 送信元 ID、ドライバーによって異なる
    buf[1]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[2]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[3]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[4]=0;                                     // センサ値
    buf[5]=1;                                     //角部センサ
    buf[6]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[7]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    //(送信先 id, フレーム状態（0 が標準）, データの長さ[Byte], 送信データ)
    CAN0.sendMsgBuf(1, 0, 8, buf);

    s1=false;
  }
  //Serial.print(s);
}
// フォトセンサ 2 の反応時に CAN 通信で中継器に送信
if (sens2 == true)
{
  if(s2 == false)
  {
    Serial.println("sencer2 ON!");

    //送信データの作成
    buf[0]=sid;                                   // 送信元 ID、ドライバーによって異なる
    buf[1]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[2]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[3]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[4]=1;                                     // センサ値
    buf[5]=0;                                     // 中心部センサ
    buf[6]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[7]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    //(送信先 id, フレーム状態（0 が標準）, データの長さ[Byte], 送信データ)
    CAN0.sendMsgBuf(1, 0, 8, buf);
    s2=true;
  }
  //Serial.print(s);
}
else
{
  if(s2 == true)
  {
    Serial.print("sencer2 OFF アドレス:");
    Serial.println(sid);

    //送信データの作成
    buf[0]=sid;                                   // 送信元 ID、ドライバーによって異なる
    buf[1]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[2]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[3]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[4]=0;                                     // センサ値
    buf[5]=0;                                     // 中心部センサ
    buf[6]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    buf[7]=0;                                     // ﾘｻﾞｰﾌﾞ　未使用
    //(送信先 id, フレーム状態（0 が標準）, データの長さ[Byte], 送信データ)
```

```
            CAN0.sendMsgBuf(1, 0, 8, buf);
            s2=false;
          }
          //Serial.print(s);
        }
        delay(100);
}
```

# 謝辞

本研究は高知工科大学大学院基盤工学専攻知能機械工学コース、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。

本研究を行うにあたり、川原村敏幸教授とはよく議論を交わし、多くのアイデアを頂きました。研究当初、制御や電子分野で知識不足だった時に、十分に勉強に集中する時間と環境を頂き、優しく丁寧にご指導してくださったおかげで、学ぶことの楽しさに改めて気づき、本研究を進めることが出来ました。心より感謝申し上げます。

山本利水先生には外部講師として月 2 回当研究室に訪問していただき、制御に関する基礎知識を教わり、研究に行き詰まったときにはアドバイスを頂きました。また、プライベートでも船釣りなどの遊びに誘っていただき、とても充実した大学生活を送ることができました。本当にありがとうございました。

博士課程である安岡達也さんには、研究の方針や発表資料の作り方、論文を書く際の細かなルールなど研究に関わることは勿論のこと、最新のニュースや日常の豆知識に至るまで、様々な分野に関してアドバイスや情報を教えていただき、その豊富な知識に何度も助けられました。本当にありがとうございました。

当研究室に所属の同期である小林俊介君は、研究で使う部屋が同じ A157 である上、バイト先も同じであり一番交流が多く、研究から日常のことに至るまで様々な意見交換をしました。その分意見の違いから衝突することもありましたが、自分とは視点の違う考え方を知るいい機会となりました。小松正彦君は、実験や資料をまとめるために研究室で一生懸命作業をする姿をよく見かけ、自分も負けていられないなと努力をするきっかけになりました。また、初めて学会に参加したときには経験者として先導してもらい、とても心強かったです。須佐美大夢君は、同期とは思えないほど知識も経験も豊富で、高い計画性や柔軟な対応力など見習うべき点も多く、最も身近な模範生としてこっそり参考にさせてもらっていました。３人とも自分とは研究分野は違うものの、お互い親身になって意見を交わし、切磋琢磨しながら大学院の 2 年間とても有意義に研究に向き合えたと思います。本当に感謝しています。

同研究室の後輩である M1 の荒木穂孝君には、研究会では積極的に質問や意見をしてもらい、理解をしてくれようとする姿勢がとてもうれしく、研究会で発表をすることへのモチベーションになりました。岡田裕明君は、よく研究のために利用していた部屋の A157 で最も会う機会が多く、お互いの研究の進捗や何気ない日常会話を交わし、この修士論文を執筆する際にもよく相談をしたりし、気持ち的な面でとても頼りにしていました。あと、一番感謝していることは大量に釣ってきた魚を捌いてくれたことです。梶亮介君には、研究で同じ Python のプログラムを使っていることから、研究時によくわからないエラーを吐いた時など、よく相談に乗ってもらっていました。また、趣味嗜好が似ていることもあり、プライベートでも遊んだりして、良き友人であったと思っています。３人ともとても頼りになる後輩で、自分の方が先輩なのによく支えられていたと思います。本当にありがとうございました。

B4 の宇佐美忠信君、大川我覚君、岡田達樹君、田上遼君、B3 の大橋亮介君、中村祐輔君、水元圭君、法川隼人君には研究会でたくさんの意見を頂き、日常生活でも親身に接してくれたおかげで、楽しく充実した研究室生活を送ることが出来ました。ありがとうございます。最後に、講義を通じて研究への指針を示して下さった高知工科大学の先生方、大学生活を共に過ごした友人、そしていつでも自分の進む道を優しく見守って頂いた両親に感謝の意を表して謝辞と致します。