2022(令和 4) 年度　　修士学位論文

# 全方向搬送装置ユークリーターの制御システムの開発

# Development of control system for Euclitor, an omnidirectional transport device

2022 年 9 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

1235095　和仁原 季也

指導教員： 川原村 敏幸 教授

# 目 次

# 第1章　序論

## 1.1　背景

### 1.1.1　移動手段の歴史

人類は文明を発達させるとともに様々な移動手段を考案してきた。紀元前 1 万年頃には船の原型となるものがあったとされ、歴史上最も古い乗り物の一つとされている。その後、紀元前 3000 年頃には風の力を使う帆のついた船が使われるようになったと言われている。同じ頃、陸上では車輪が発明され、馬やロバが車を引くことで徒歩と比較して長距離を短時間で移動できるようになった。

風や動物など自然の力を利用する乗り物の時代は長く続いたが 18 世紀半ばに蒸気機関が発明され、人類は機械による動力を手にすることができた。第一次産業革命としてよく知られている出来事である。ボイラーで発生させた水蒸気を動力とする蒸気機関車は、それまでの乗り物と比較して格段に多くの人や荷物を運べるようになり、あらゆる産業を大きく発展させる鍵となった。

19 世紀後半には電気や、石油を燃料とする内燃機関が発明され、人類は新たなエネルギー源を手にした。第二次産業革命である。ダイムラーやベンツ、フォードなどにより自動車が発明され、人々は駅や運行ダイヤの制限を受けることなく、好きな時に好きな場所へ移動できるようになった。また、ライト兄弟による航空機の有人動力飛行に成功し、ついに人類は空の旅へと漕ぎ出した。

20 世紀後半にはジェットエンジンを搭載した大型の旅客機が開発されるなど、人類の移動手段はより大型に、より高速に、より長距離を移動できるように進化してきた。現代社会では自家用車や大型バス、トラック、鉄道に加え、飛行機や大型輸送船など陸海空にわたり様々な移動手段が使用されている。[1]

### 1.1.2　問題提起

上述のように、各種移動手段は多くの人々が自由に移動できる便利な社会を実現し様々な産業を発展させてきたが、一方で環境問題や安全性、エネルギーの問題など大きな社会課題を引き起こした。

また、重厚長大的な進歩に注目してきたことにより、近場や密集地での移動手段はあまり開発されていない (図 1.1)。空港やショッピングモール、物流倉庫などに着目すると動く歩道やエスカレータ、ベルトコンベアなどが存在するが、これらは決められたレールの上しか搬送させることができず自由度が低いという欠点がある。

さらに、現行の移動手段の大半は他点起発事象予測方式である (図 1.2(a))。すなわち運転者自身が進路上で起こり得る事象を予測しながら危険を回避する必要があり、これらの予測には限界があることから事故を完全に無くすことは困難である。

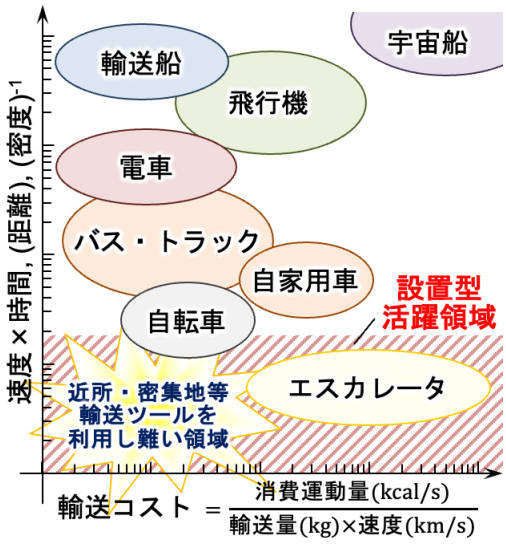



図 1.1 各種移動手段の活躍領域

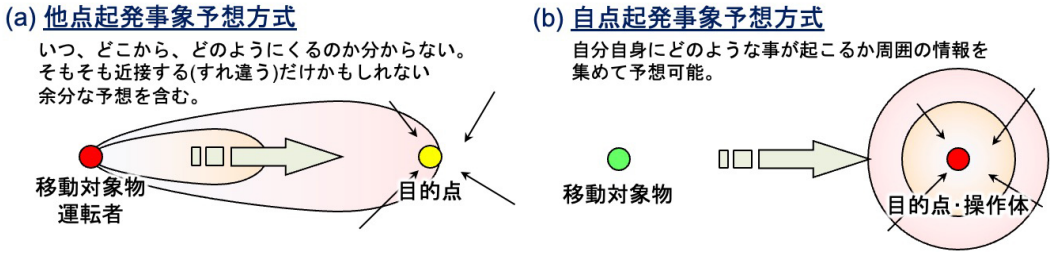



図 1.2 他点および自点起発事象予測の違いに関する概念図

### 1.1.3 次世代の移動手段の提案

我々の研究室では上述の問題を解決する次世代の移動手段として「ユークリーター ®[*1]」の開発を行っている (図 1.3)。

ユークリーターとは概括すると動く歩道を全方向に搬送できるように拡張したシステムであり、床一面にデバイスを敷き詰め、それらが個別に作動することで上に乗った物や人を搬送させるシステムのことである。

ユークリーターを用いて移動する場合、敷設されたデバイス自身が周囲の情報を集めて制御することができるため、搬送対象物に起こる事象を予測することができ、現行の移動手段と比較して格段に高い安全性を確保する事ができる (図 1.2(b))。

[*1]商標登録 第 6313573 号, 第 6313574 号

また、消費エネルギーの観点からも、現行の動く歩道やベルトコンベアでは搬送物が載っていない部分、すなわち搬送に寄与しない部分も含め装置全体を駆動させる必要があるため消費エネルギーが大きく効率が悪い。対してユークリーターでは搬送物直下のデバイスのみを選択的に駆動させる事ができるため消費エネルギーを最小限に抑える事ができ、搬送効率を抜本的に改善できる。


図 1.3 ユークリーターのコンセプト

## 1.2 ユークリーター

### 1.2.1 概要

ユークリーター (Euclitor) とは、球体を保持しつつ全方向に回転させることができる機構を有した装置を床一面に敷き詰めることにより、搬送物をどこからでも自由自在に搬送させることができる自律分散型のシステムである。

このシステムの土台となったのは映画「ドラえもん のび太と銀河超特急」に登場するベアリングロードと呼ばれる秘密道具である。地面一面に球体 (ベアリング) が敷き詰められており、それらが自在に回転することで上に乗った人や物を搬送することができる。ユークリーターのコンセプトはこのベアリングロードに着想を得て構築された。

また、ユークリーターという名称は以下の 2 つの由来により決定された。

1. 直線や平面、空間などの概念について体系立てたユークリッド幾何学 (Euclidean geometry)
2. エスカレーターはラテン語で階段を意味する「scala」に接頭語の「e」、人や機械、物などを示す接尾語である「or」を加えた造語

ユークリーターの構想は図 1.3 に示す第 1 世代 (G1) から第 4 世代 (G4) に及ぶ。まずは第 1 世代として、倉庫や市場などでの荷物の搬送を行うシステムを実装することを目標とする。次に第 2

世代では病院や介護施設などの限定された屋内空間において、移動が困難な人の歩行支援を行う。第 3 世代ではさらに適用面積を広げて、ショッピングモールや空港など、歩いて移動するには少し苦労を感じる距離の移動を支援する。第 4 世代では屋内にとどまらず屋外においてもあらゆる人や物の搬送を支援するシステムを目指す。

### 1.2.2　先行研究 (機構系開発)

2015 年ごろからユークリーターの実現に向けて、球体を用いた全方向搬送装置の筐体設計が行われてきた。球体型の筐体は大きく分けて第 1 世代から第 3 世代まで開発されており、運用されている。

また、球体型第 2 世代の機構から枝分かれする形で、より伝達効率を高めることを目的として、ベルト型の機構の開発も行われてきた。ベルト型は現在第 3 世代まで開発されており、こちらも並行して運用されている。

以下にそれぞれの世代における機構について概要を述べる。

#### (1)　球体型第 1 世代

球体型第 1 世代はユークリーター実現に向けたコンセプトの検討として開発された。ドラえもんに登場するベアリングロードでは球体自身が単体で自在に回転することで人の搬送を行っているが、小さな球体の内部にセンサや駆動用の動力源を収めることは現実には困難であるため、図 1.4 に示すような 2 段式構造とすることで 1 段目の球体群を制御する機構とした。

藤川、吉本により製作された球体型第 1 世代の外観図を図 1.5 に示す。[2][3] 駆動系のアクチュエータは搭載されず、手で 2 段目の球体を回転させることで 1 段目の球体群が回転するかどうかの動作実験を行い、適切な球体部の公差の検討がなされた。筐体の強度については佐藤により構造解析が行われ、十分な強度を有していることが報告されている。[4]

また、藤川、秋山、竹中により駆動系の検討や複数配置のモータで球体の回転方向を制御するための運動学が報告されている。[2][5][6]

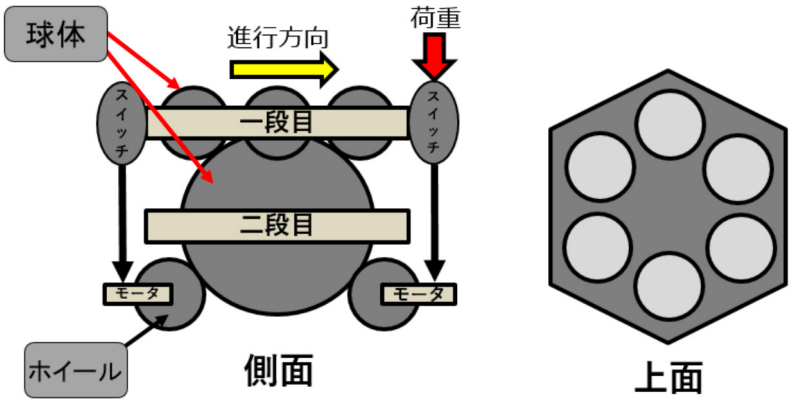
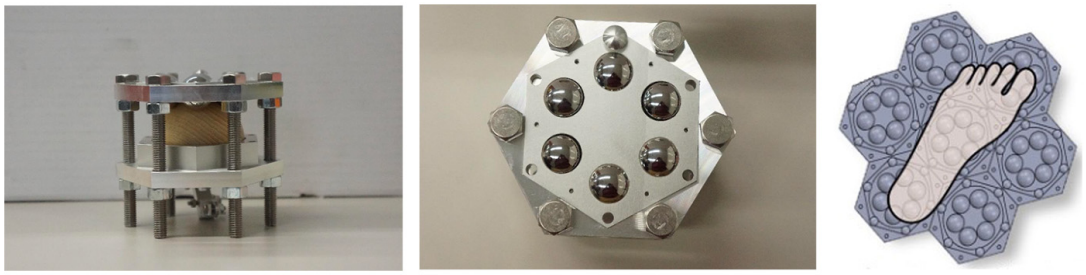



図 1.4 球体型第 1 世代のコンセプト



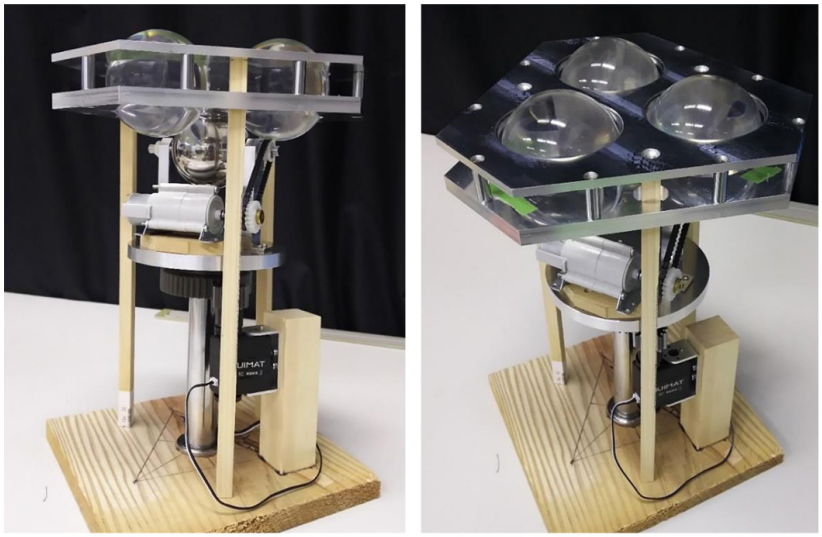
(a) 側面図　(b) 上面図　(c) 敷詰め時

図 1.5 球体型第 1 世代

### (2)　球体型第 2 世代

球体型第 2 世代では 1 段目の球体の保持方法を面接触からボールプランジャによる点接触に改良し、摺動抵抗の低減を図った。また、2 段目の球体を駆動させるための DC モータを追加し、サーボモータにより 2 段目テーブルを回転させることで方位決定を行う機構とした。駆動モータと方位決定のサーボモータの 2 つに分けることにより使用アクチュエータを減らすことができ、複数配置のモータによる制御と比較して制御を簡略化することができた。

また、動作試験を行い、軽負荷状態では 1 段目と 2 段目の球体の間で滑りが見られたが、1.35kg の荷物の搬送が可能であることが狩野、鈴鹿により報告されている。[7][8]



(a) 側面図　(b) 上面図

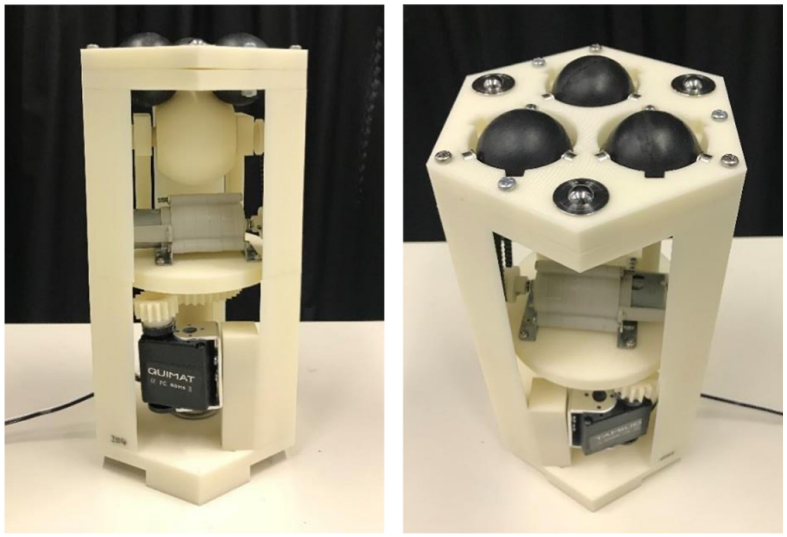
図 1.6 球体型第 2 世代

### (3)　球体型第 3 世代

球体型第 3 世代では基本機構は第 2 世代を踏襲しつつ、筐体を正六角柱状に変更し小型化することで敷詰め密度を向上させた。また、1 段目の球体をアクリル球からゴム球へ変更し、摩擦力

を向上させることで駆動力の滑りを抑えた。

動作試験では 2.5kg までの荷物が搬送可能であることが長嶋により報告されており、第 2 世代と比較して 1.9 倍の搬送能力の向上を実現した。[9]



(a) 側面図　　(b) 上面図

図 1.7 球体型第 3 世代

### (4)　球体型第 3 世代改良型

球体型第 3 世代での動作試験で球体間の動力の伝達や方位決定に問題がなかったことを受け、量産に向けた改良がなされた。本改良型では方位決定用のサーボモータを回転テーブル上に移設することで駆動・方位決定機構部を短縮することができ、筐体全体の小型化ができた。また、製造を外注し、筐体全般の素材を 3D プリンタによる ABS からアルミ合金に変更することで強度と精度の向上を行った。

動作試験では 3.0kg の搬送が可能であり、条件によっては 9.0kg の搬送にも成功したことが狩野により報告されている。[10]

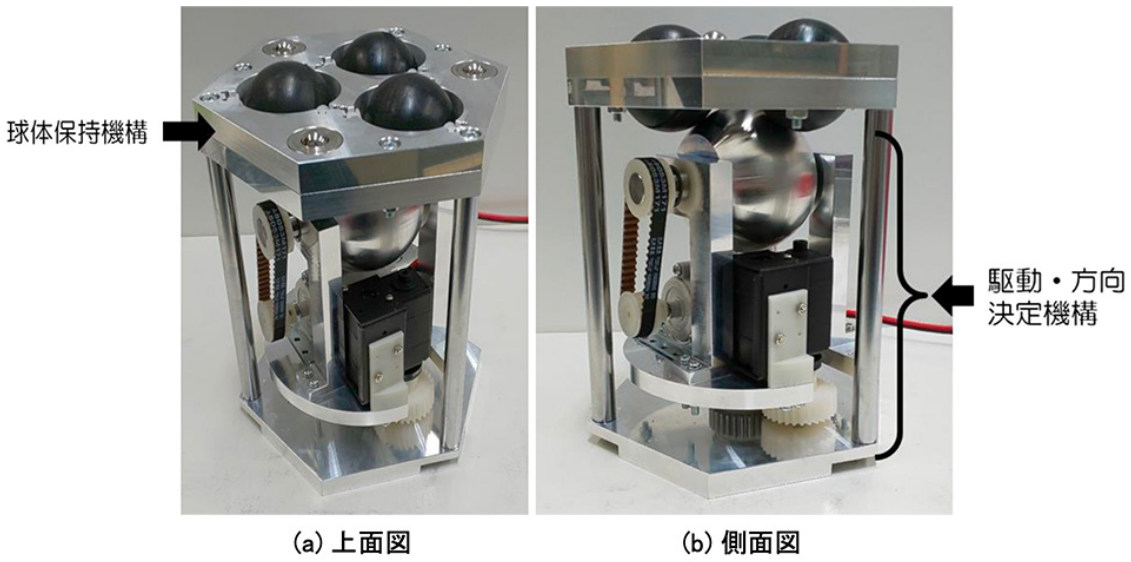



図 1.8 球体型第 3 世代改良型

### (5)　ベルト型第 1 世代

球体型第 2 世代でサーボモータによる方位決定機構を確立したことにより、1 段目の搬送物と接触する面は球体でなくとも成立することに気がついた。そこで伝達効率の向上を目的として、搬送部にベルトを用いた機構の開発が竹中、石井により行われてきた。[11][12]

図 1.9 に示すベルト型第 1 世代の筐体はベルトを最密に配置するため、円形となっているが敷詰め時には六角柱状の外枠を取り付けることで隙間なく敷詰めるようにする。

動作試験では 2.5kg までの搬送が可能であることが報告されている。

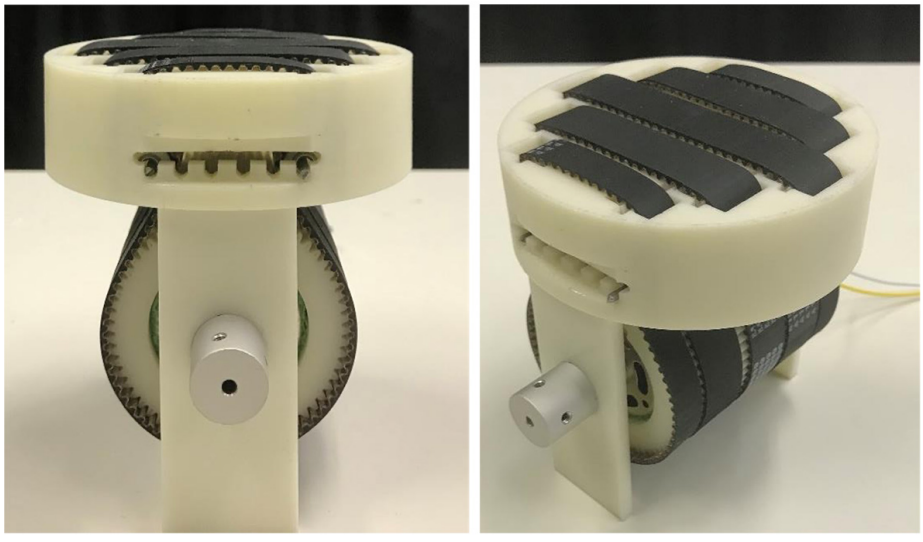
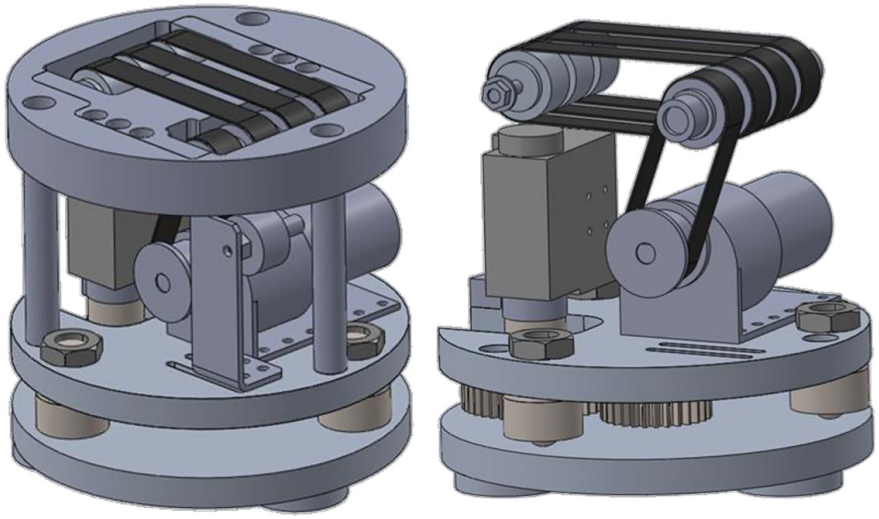



図 1.9 ベルト型第 1 世代

### (6)　ベルト型第 2 世代

ベルト型第 2 世代ではベルトの駆動方式をセンタードライブ方式からヘッドドライブ方式に変更し、球体型第 3 世代改良型と同様に方位決定用のサーボモータをテーブル上に移設することで筐体の高さを短縮した。これにより筐体全体の剛性が向上し、ベルト型第 1 世代で課題となっていた駆動時の歪みを抑制することができ駆動損失を低減することができた。

ヘッドドライブ方式では逆回転を行うと搬送面がベルトの緩み側となり、有効張力が低下するため一般的には非推奨とされているが、動作試験では両回転とも 5.0kg の搬送が可能でありベルト型第 1 世代と比較して 2 倍の搬送能力を有していることが石井により報告されている。[13]



(a) 全体図　　(b) 駆動部詳細図

図 1.10 ベルト型第 2 世代

### (7)　ベルト型第 3 世代

ベルト型第 2 世代で良好な搬送能力が得られたことを受け、ベルト型第 3 世代ではベルトの占有面積を拡大する改良を行った (図 1.11)。また、ヘッドドライブの駆動軸を中心に配置することでベルトの張り側と緩み側が互い違いになるようにし、駆動方向による搬送能力の差を緩和した。

動作試験では 7.0kg までの搬送が可能であり、ベルト型第 2 世代より更なる搬送能力の向上を実現した。[13]

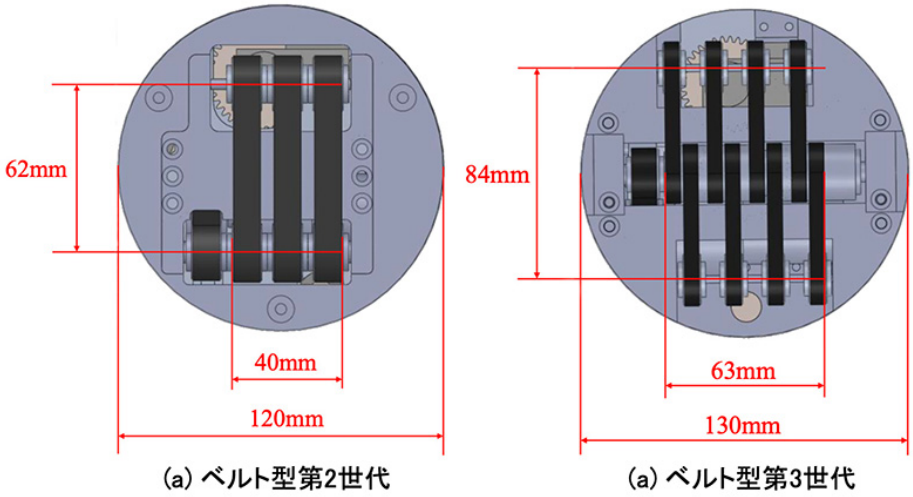



図 1.11 ベルト占有面積の比較

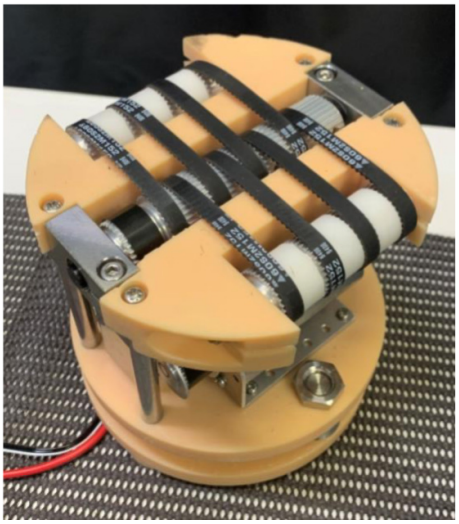


図 1.12 ベルト型第 3 世代

### 1.2.3　先行研究 (制御系開発)

機構系の開発の進展に並行し、2018 年ごろから制御系および通信系の開発が鈴鹿により行われてきた。[14]

#### (1)　制御基板の開発

DC モータとサーボモータを同時に制御するため作成された基板を図 1.13 に示す。無線通信による制御システムを構築する構想があったため、Wi-Fi モジュールの ESP-WROOM02 を搭載している。モータの制御は可変抵抗の値により PWM のデューティ比を制御する方式で行っている。

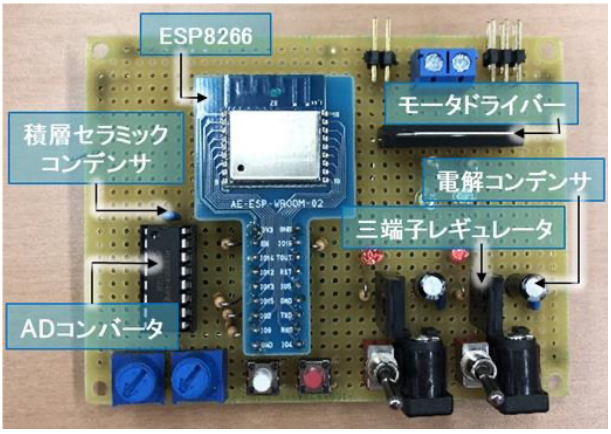


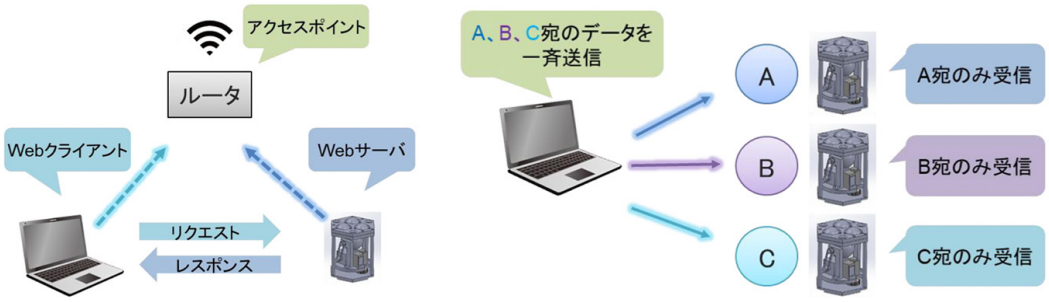

図 1.13 制御基板 v1

### (2) 無線通信による制御システムの構築

無線通信での制御についてはまず Wi-Fi を用いた Web サーバ方式が構築された (図 1.14(a))。動作試験ではユニット 1 台分の制御が可能であることが確認されたが、多数のユニットの制御を行うためにはユニットごとに Web クライアントが必要となり、プログラム量が膨大となり動作が遅くなることが予期されたため、Web サーバ方式での開発は見送られた。

次に候補となったのが UDP(User Datagram Protocol) 通信による制御である (図 1.14(b))。UDP 通信は送信側がデータを一斉送信し、受信側が自分宛のデータのみ受信するという通信方式である。送信側は受信できたかどうかの確認を行わないため、多少のデータ欠損が生じる可能性があるが、スピード重視で大きなデータを処理できるため採用した。

ホストとなる PC とクライアントとなる基板間での要素試験では問題なく通信ができていたが、モータの制御パラメータを送信するとクライアント側でリセットがかかり、モータが停止するという問題が生じた。原因究明の結果、Wi-Fi ルータや部屋を変更すると動作しなくなることがわかり、相性問題や電波干渉による問題と推察された。ユークリーターは工場などノイズの多い環境で使用されることが想定されるため、環境に左右される無線通信での制御システムの構築は困難と結論づけ、開発を一時中断することとした。




(a) Wi-FiによるWebサーバ方式での制御方法　　(b) UDP(User Datagram Protocol)通信での制御方法

図 1.14 無線通信による制御システムの構築

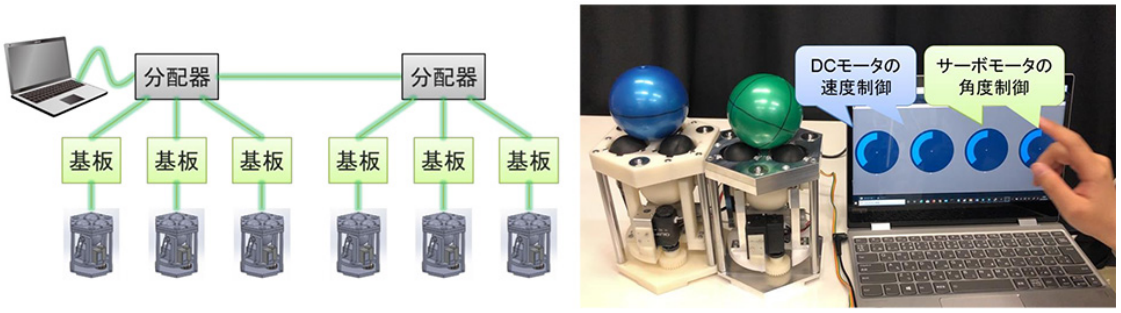
### (3)　有線通信による制御システムの構築

有線通信はノイズの多い環境下でもほとんど影響を受けないため、無線通信に比べ優位であると考え、有線通信での制御へと方針を切り替えた。そこでまず通信方式には少ない配線で通信ができるシリアル通信を採用した (図 1.15(a))。シリアル通信の出力側には Processing を使用し、画面に表示されたノブを操作してモータの出力値を変更するプログラムが作成された (図 1.15(b))。

動作試験では正常にモータの制御ができていることが確認され、1 つの基板で 2 つのユニットの個別制御が可能であることが実証された。この結果を受け、接続ユニット数を増やしての動作確認へと移る予定であったが、この制御方式ではユニットを接続した順番にしか制御ができないことが分かった。すなわちユニットをランダムに制御することができないため、ユークリーターが目指す搬送物直下のユニットのみを選択的に制御することが困難であることが分かった。

また、分配器を用いて接続数を増やすため一部で断線などにより通信に問題が生じた場合、それ以降のユニットが制御不能になるという問題も発覚した。

これらの問題に対処するため、シリアル通信による制御システムの開発は中止し、他の手段による制御システムの検討を行った。




(a) シリアル通信を用いた制御システムの概要　　(b) シリアル通信の動作実験

図 1.15 シリアル通信による制御システムの構築

### (4)　CAN 通信による制御システムの構築

前述の問題を解決する手段として、CAN(Controller Area Network) 通信を用いた制御システムの構築が行われた。CAN 通信は通信線であるバスと接続機器のノードにより構成されるバス通信の一種であり、バスにノードを接続すればネットワークを構築することができるためノードの追加や削除が行いやすい。そのため接続するユニットの個数や配置を変更しやすく、ネットワークの設定を変更することなく故障したユニットのみを取り替えることができるため保守性にも優れており、ユークリーターの制御系に最適であると判断した。

また、バスに空きがあればどのノードからも双方向に通信を開始することができるため、今後ユニットにセンサなどが追加された際にもフィードバックが可能である。この観点から、1 つの CAN 基板に複数のユニットを接続する方式ではなく、1 つの CAN 基板に 1 つずつのユニットを接続するシステムとした (図 1.16)。

Processing のスライダを用いてモータを制御するプログラムを作成し動作試験を行った結果、9 個のユニットを個別に制御することができた。

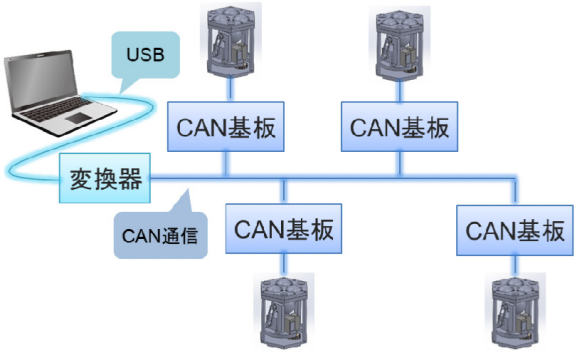



図 1.16 CAN 通信による制御システムの構築


図 1.17 CAN 通信による動作試験

### 1.2.4　先行研究 (付加機能の追加)

その他の開発として、ユークリーターの駆動電力の自給自足化を目的とした太陽電池の設置やフィードバック系としてユニットへの赤外線センサの設置、赤外線センサを活用したルート決定アルゴリズムの検討などが高井、亀岡、澤田により報告されている。[15][16][17]

## 1.3　本研究の目的

本研究の目的は、複数のユークリーターをアレイ化したシステムを構築し、それぞれを統合制御することにより搬送物を目的地に搬送するシステムを開発することである。これによってまずは第 1 世代である物流業界での荷物搬送の実用化を目指す。

## 1.4　類似の研究

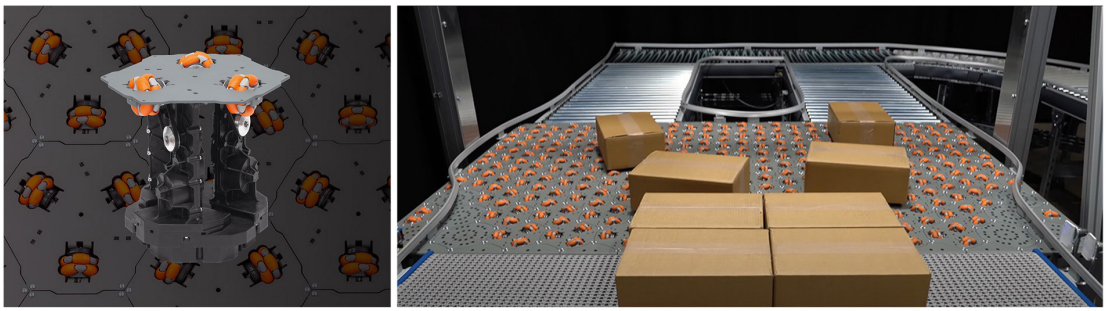
類似の研究として cellumation 社の celluveyor や Beckhoff Automation 社の XPlanar、九州工業大学の球体駆動式全方向移動機構などが挙げられる。

それぞれの特徴および本研究との相違点を以下の項で述べる。

### 1.4.1　celluveyor

celluveyor はオムニホイールを用いた全方向搬送装置である。[18] ユークリーターと同様に六角柱状の筐体を床一面に敷き詰めて搬送を行う。筐体の上面には 3 つのオムニホイールが搭載されており、それぞれのオムニホイールを個別に駆動させることで搬送方向を決定する (図 1.18(a))。

celluveyor は既に実用化のレベルまで開発が進んでおり、ドイツの輸送機関 DHL の倉庫で搬送試験が行われている。cellumation 社の Web ページ[*2]で公開されている動画では celluveyor を用いて荷物を別々のベルトコンベアへと導く仕分け作業や、荷物の箱を回転して整列させる様子が紹介されている (図 1.18(b))。



(a) ユニット　　(b) 搬送試験の様子

図 1.18 celluveyor の概要

[*2] https://cellumation.com

一見、本研究のユークリーターと類似点が多く、ユークリーターが後追いのように見えるかもしれないが、celluveyor にも短所がある。それはオムニホイールを使用しているため、非平坦面との動力伝達に限界がある点である。つまり、底面が平らな箱型の荷物以外の搬送には不向きであり物流用途からの発展性に限界があるという点である。ユークリーターでは最終的に人の歩行支援を目標としているため、この点で本研究の独自性がある。

また、celluveyor では 3 つのオムニホイールを個別に制御するため、動力源となるモータが少なくとも 3 つ必要となる。対してユークリーターでは方向決定用のサーボモータと駆動用の DC モータの 2 つで制御することができるため、故障に対する信頼性やコスト面での優位性があると考えられる。

一方で動画内ではメンテナンスの方法や、荷物の位置を取得するためにカメラを使用していることなどが紹介されており、本研究において参考にすべき点は多い (図 1.19)。

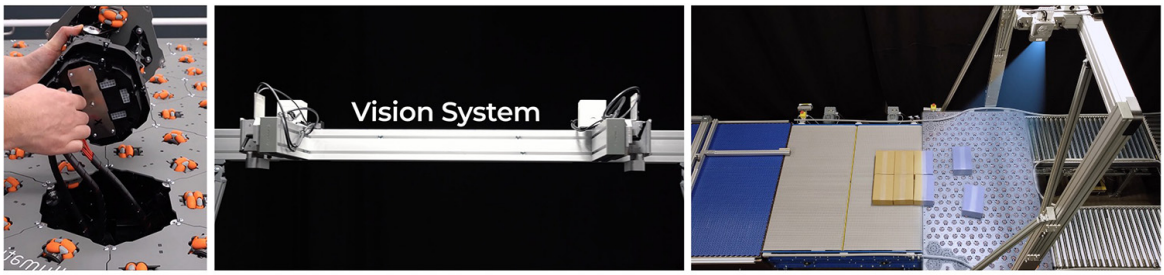



図 1.19 celluveyor の紹介動画の抜粋

### 1.4.2 XPlanar

XPlanar は床面に仕込まれた制御装置の上を電磁力により浮遊させたタイル状の筐体を全方向移動させることで搬送を行うシステムである。[19] 平面移動に加え、タイルをチルトさせることも可能で高い自由度での搬送を実現している (図 1.20)。

非接触で浮遊して搬送させるため、搬送物への衝撃がなく繊細な荷物の搬送も可能である点は XPlanar でしか実現できない利点ではあるが、現時点では搬送可能重量に制約があるため重量物の搬送は困難である。また、磁気に敏感な物は直接乗せて搬送することができない可能性がある。

加えて、搬送を行うためにシステム本体に加えて荷物を乗せる筐体が必要不可欠である点がユークリーターや celluveyor と大きく異なる点である。ユークリーターや celluveyor では荷物を乗せる筐体は必要とせず、システムの上を搬送させることができるが、XPlanar では荷物を筐体に乗せる操作を人力または機械によって行う必要があり、自律的な搬送への障壁が高いと考えられる。

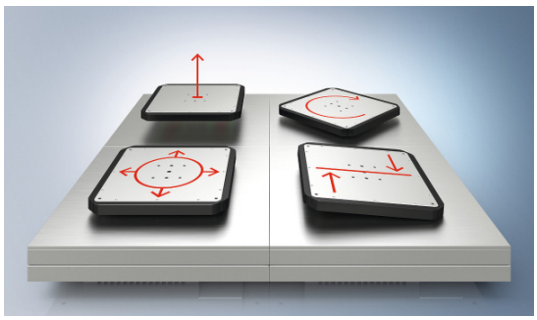
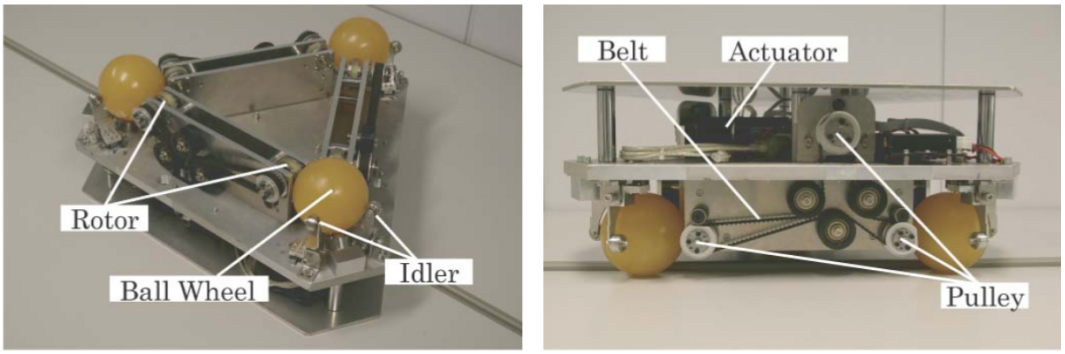


図 1.20 XPlanar の搬送自由度

### 1.4.3 球体駆動式全方向移動機構

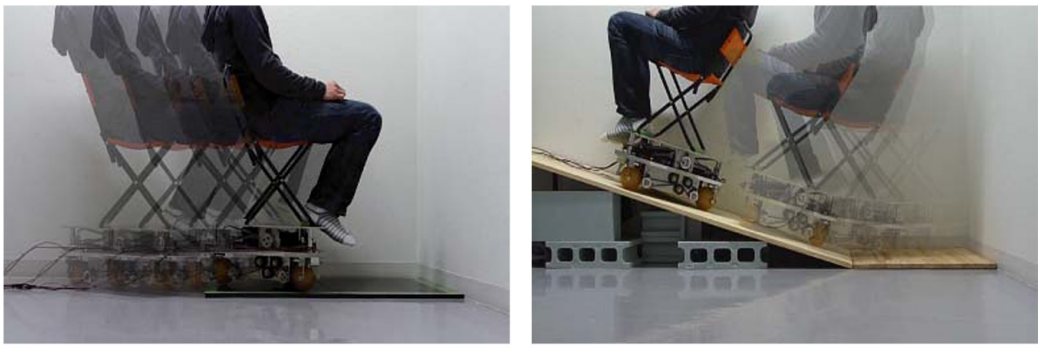
球体を用いた全方向移動機構が九州工業大学の石田らにより報告されている。[20] 図 1.21 に示すように車両に 3 つの球体が取り付けられており、各球体は 2 つのアクチュエータにより駆動される。走行試験として段差乗越え、溝踏破、斜面登坂などが行われており、成人が乗車した状態で屋内利用に十分な性能が実証されている。

球体を用いた機構ではあるが、筐体に乗り込んで移動するという点でユークリーターとは異なる技術開発である。しかし、球体の駆動方向に関する運動学の導出や走行性能など本研究において参考にすべき点は多い。




(a) Perspective View　　(b) Side View

図 1.21 球体駆動全方向移動機構の外観



(a) 段差乗越え運動　　(b) 坂道登坂運動

図 1.22 走行試験の様子

### 1.4.4　類似研究との性能比較

類似の搬送装置との性能の比較を表 1.1 に示す。比較項目については以下の通りである。

| | |
|---|---|
| 搬送速度 | 1 分間に何 m 搬送可能か |
| 搬送荷重 | 搬送長 1m あたり、搬送面積 1m$^2$ あたり何 kg まで搬送可能か |
| 搬送用筐体必要性 | 搬送の際に荷物を乗せる筐体が必要か |
| ユニット間連動性 | ユニット同士が連動しているか |
| 輸送可能次元数 | 何次元に搬送可能か |
| 各ユニット独立駆動 | 各ユニットが独立して搬送しているか |
| 電源自律分散化 | 電源が分散しているか |

実用化必要水準は物の搬送を対象とした場合を考えており、現行のベルトコンベアなどの能力から推定した値である。

表の通り、現在のユークリーターではまだ必要水準を満たしていないが、ユニット間連動性については本研究によって改善を行い、搬送速度や搬送荷重、電源自律分散化についても並行して開発を進めているため、数年以内に実用化水準を満たすものと考えている。

表 1.1 類似の搬送装置との性能比較

| 比較項目 | 現行<br>球体型 | 現行<br>ベルト型 | 実用化<br>必要水準 | ベルトコンベア<br>(IHI 製 SCU 型) | celluveyor<br>(cellumation) | XPlanar<br>(Beckhoff<br>Automation) | TDAMNEW<br>(土佐電子) |
|---|---|---|---|---|---|---|---|
| 搬送速度 (m/min) | ~3.0 | 10~14 | 20 | 3~25 | ≈2 | ≈2 | 20 |
| 搬送荷重 (kg/m) | ~40 | -60 | 100 | 150 | ≈100 | ≈15 | 100 |
| 搬送荷重 (kg/m$^2$) | ~330 | ~620 | 1000 | − | ~1800 | ~76 | − |
| 搬送用筐体必要性 | 無 | 無 | − | 無 | 無 | 有 | 有 |
| ユニット間連動性 | △ | △ | ◎ | ◎ | ○ | ○ | − |
| 輸送可能次元数 | 2 | 2 | 1 | 1 | 2 | 2 | 2 |
| 各ユニット独立駆動 | ○ | ○ | × | × | ○ | ○ | ○ |
| 電源自律分散化 | × | × | × | × | × | × | ○ |

## 1.5 本論文の構成

本論文の構成について述べる。

第 1 章では本研究室で開発している新たな移動手段である「ユークリーター」のコンセプトおよび開発動向を示し、本研究の目的について述べた。

第 2 章ではアレイ化に伴う諸種の問題への事前検討について述べる。

第 3 章ではアレイ化したユークリーターの状態をコンピュータ上で再現するためのシミュレータの開発について述べる。

第 4 章では Python を用いたグラフィカルな 1 対 1 コントローラの開発について述べる。

第 5 章では CAN 通信を用いた複数ユニットの統合制御を行うコントローラの開発について述べる。

第 6 章では本研究の成果と今後の展望について述べる。

また、付録として作成したプログラムのソースコードを添付する。

# 第2章　アレイ化に向けた事前検討

ユークリーターのアレイ化に際して 2 つの検討課題がある。1 つは敷詰め時のユニットへのアドレスの割り付けの方法である。もう 1 つは膨大な数になることが予期される制御配線系についてである。

## 2.1　各ユニットへのアドレスの割り振り

ユニットの外形が六角形であるため、敷詰め時のユニットへのアドレス (位置情報) を考える際に不都合が生じる。一般に使用される直交座標系での敷詰め時のアドレス割り振りを図 2.1 に示す。

図のように隣り合うユニット間の距離を 1 とすると、2 行目のユニットの $x$ 座標はユニット半個分右にオフセットし、座標が整数にならないためアドレスの割り振りには適さない。また、$y$ 方向には $\sqrt{3}/2$ ずつ配置されるため、直感的に座標と配置が結びつきにくいという問題がある。

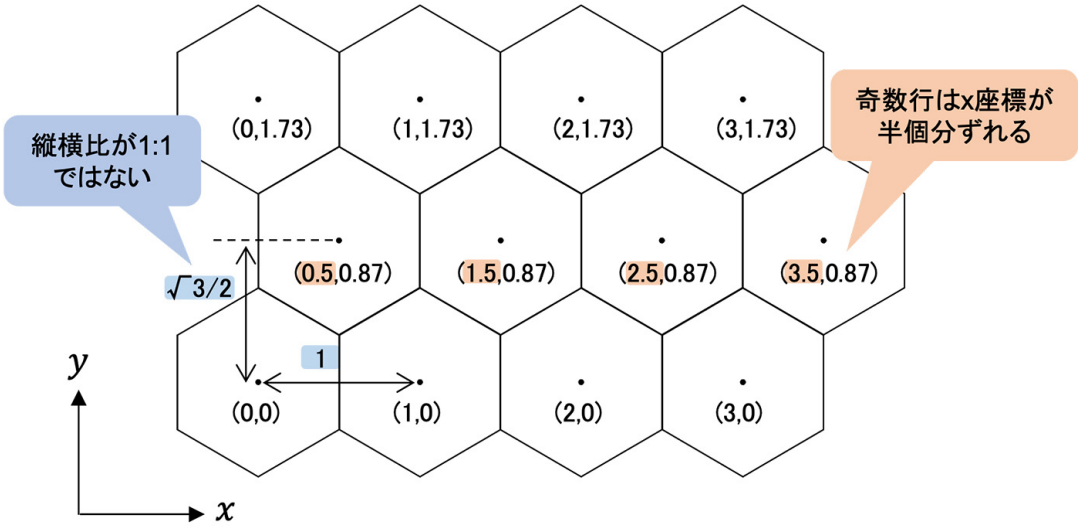



図 2.1 直交座標系でのアドレス割り付けの問題点

### 2.1.1　斜交座標系の導入

上述の問題を解決するため、斜交座標系を使用する。斜交座標は $x$ 軸と $y$ 軸が斜めに交わった座標系であり、平面幾何学や気象学における上空の気温プロファイルのグラフなどでよく使用される座標系である。また、ユークリーターと同様に六角形のセルを有するグラフェンやカーボンナノチューブなどの座標系でもよく用いられている。[21][22]

ここでは $x$ 軸と $y$ 軸が 60° で交差する 60° 斜交座標系について考える。図 2.2 に示すように 60° 斜交座標系を使用すると問題となっていた 2 行目の $x$ 座標は整数となるため、アドレスとして扱いやすく座標と配置が結びつきやすくなる。

一方で、斜交座標系における座標を用いて偏角や距離を求める場合には見た目の (直交座標系での) 偏角や距離と異なるため、座標変換が必要となる。例えば斜交座標系において、原点 (0, 0) から (1, 1) へ向かうベクトルの偏角は 45°、ノルムは $\sqrt{2}$ となるが、直交座標系では原点 (0, 0) から (1.5, $\sqrt{3}/2$) へのベクトルとなり、偏角は 23.4°、ノルムは 1.73 となる。

このように斜交座標系を用いることによるデメリットもあるが、座標変換を行なってから偏角や距離を求める機能をプログラムに実装することは簡単であるため大きな問題にはならず、それ以上にユニットの配置と座標が直感的に結びつきやすいというメリットの方が大きいと判断し、斜交座標系を採用した。

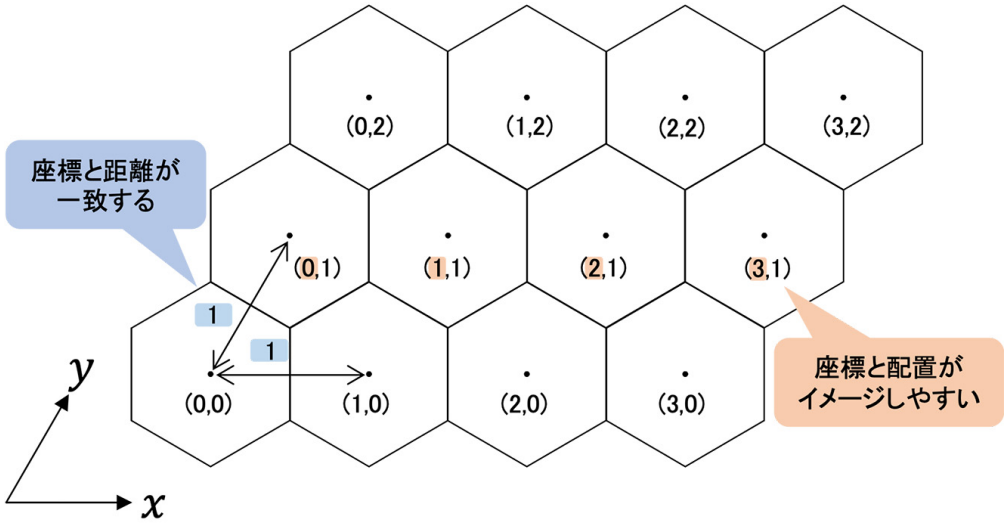



図 2.2 60° 斜交座標系でのアドレス割り付け

### 2.1.2 座標変換について

直交座標系と斜交座標系間での座標変換について述べる。図 2.3 に示すように $X$-$Y$ 軸からなる斜交座標系と $X'$-$Y'$ 軸からなる直交座標系について、$X'$ 軸と $X$ 軸がなす角を $\theta$、$X'$ 軸と $Y$ 軸がなす角を $\phi$ とする。ここで、斜交座標系における点 $P$ の座標は $(x, y)$、直交座標系における点 $P$ の座標は $(x', y')$ である。

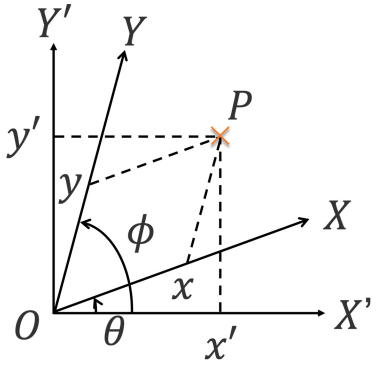



図 2.3 座標変換

**斜交座標系から直交座標系への座標変換**

斜交座標系 $(x, y)$ から直交座標系 $(x', y')$ への座標変換は以下となる。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \cos\phi \\ \sin\theta & \sin\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.1}$$

ここで、$X$ 軸と $X'$ 軸は共有するため、$\theta = 0°, \phi = 60°$ とすると、

$$\begin{aligned} x' &= x\cos 0° + y\cos 60° \\ y' &= x\sin 0° + y\sin 60° \end{aligned} \tag{2.2}$$

よって、

$$\begin{aligned} x' &= x + \frac{1}{2}y \\ y' &= \frac{\sqrt{3}}{2}y \end{aligned} \tag{2.3}$$

となる。

**直交座標系から斜交座標系への座標変換**

直交座標系 $(x', y')$ から斜交座標系 $(x, y)$ への座標変換は式 (2.1) の逆行列となる。すなわち、

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{\cos\theta\sin\phi - \cos\phi\sin\theta} \begin{pmatrix} \sin\phi & -\cos\phi \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{2.4}$$

上と同様に $\theta = 0°, \phi = 60°$ とすると、

$$
\begin{aligned}
x &= \frac{1}{\cos 0^\circ \sin 60^\circ - \cos 60^\circ \sin 0^\circ} \left( x' \sin 60^\circ - y' \cos 60^\circ \right) \\
y &= \frac{1}{\cos 0^\circ \sin 60^\circ - \cos 60^\circ \sin 0^\circ} \left( -x' \sin 0^\circ + y' \cos 0^\circ \right)
\end{aligned}
\tag{2.5}
$$

よって、

$$
\begin{aligned}
x &= x' - \frac{1}{\sqrt{3}} y' \\
y &= \frac{2}{\sqrt{3}} y'
\end{aligned}
\tag{2.6}
$$

となる。

## 2.2 通信方式の検討

球体型第 3 世代改良型の外形は W120×D140×H160 である。これを 1m 四方に敷詰めた場合、8 個 ×7 個の約 60 個のユニットが敷き詰められることとなる。ユークリーターの構想の第 1 世代である、物流業界での仕分け作業などに適用することを考えるとより大きな面積にユニットを敷き詰めることとなり、敷き詰めるユニットの数は少なくとも数百から数千に、100m 四方のエリアに適用することを考えるとさらに多くのユニットが必要となり数万から数十万単位になることが考えられる。それに伴って制御配線も膨大な数となるためシンプルな配線かつ、拡張性や保守性も考慮した通信方式を採用する必要がある。

### 2.2.1 シリアル通信

シリアル通信とは 1 つの通信経路上を 1bit ずつ逐次的にデータを送ることで通信する方式のことである。対照的にパラレル通信とは複数の通信経路で同時にデータが送信される方式であり、基板内など比較的近距離で高速な通信を行う際によく利用される。パラレル通信は距離が長くなると同期が困難になり通信速度の低下や通信エラーを引き起こすため、長距離での通信には用いられない。そのため、長距離の通信やネットワークの構築にはシリアル通信が一般的に用いられる。

ユークリーターのメインマイコンに使用している Arduino ではパソコンとの通信に RS-232 をベースにした UART(Universal Asynchronous Receiver/Transmitter) というシリアル通信規格を使用している。UART では信号線として TX(Transmitter) と RX(Receiver) の 2 本のケーブルを使ってデータの通信が行われる。シリアルバスとしてよく使用される SPI や $I^2C$ とは異なり、クロック非同期式の通信方式であり、マスタやスレーブの関係のない 1 対 1 での通信方式である。そのため多数のユニットを同時に制御するには拡張性が乏しく、ユークリーターの制御系には適さない。

産業用のフィールドバスとして RS-485 という規格が存在する。RS-485 では多数対多数の通信が可能であるがその最大接続台数は 32 台であり、これもユークリーターの制御系に採用するには接続台数が不足し拡張が困難であるため採用を見送った。

### 2.2.2 EtherCAT

EtherCAT®[*3]とはパソコンなどの有線 LAN で使用されている Ethernet と互換性のあるフィールドネットワーク規格であり、産業用途で使用されることが想定されているため Ethernet より耐環境性が保証されている。

Ethernet がベースとなっているため通信速度が速く (10Mbps〜1Gbps)、リアルタイム性が高いという特徴がある。また、マスタとスレーブの関係が構築でき、接続方式にはデイジーチェーン (数珠繋ぎ) やスター型 (1 対多数)、リング型など柔軟な接続が可能である。最大接続台数は 65535 台であり、ユークリーターの制御系に適している通信方式であるが、EtherCAT に対応したコントローラが高価であるほか、Ethernet や EtherCAT 特有のプロトコルへの理解が必要であり、導入へのハードルが高いためここでは採用を見送った。

### 2.2.3 CAN 通信

CAN(Controller Area Network) 通信は自動車の制御システムなどで使用されているビークルバスの一種でマイコンやデバイスが相互に通信できるように設計されている。

通信速度は最大で 1Mbps 程度 (ケーブル長に応じて最大通信速度が制限される) であるが、デイジーチェーンによる接続が可能であり、バスの遅延時間と電気的な負荷により接続台数が制限されるが規格上は最大接続台数の制限がない。また、バスに空きがあれば全てのノードがメッセージを送信可能であり、拡張性に優れている。複数のノードが同時にメッセージの送信を開始した場合は衝突検知処理により、ID の優先度の高いノードの送信が優先される。加えて、ネットワークの設定を変更することなくノードを追加や削除が可能であり、保守性も高い。

Arduino で使用可能な CAN コントローラが安価で入手可能であり、具体的な実装方法もインターネット上に多く情報があるため、導入へのハードルはそこまで高くないと判断し、ユークリーターへの制御系として採用した。

## 2.3 結論

ユークリーターをアレイ化した際のユニットへのアドレスの割り付けは 60° 斜交座標系を用いることで解決した。

また、制御系の通信方式には CAN 通信を採用した。

[*3] EtherCAT は Beckhoff Automation GmbH の登録商標である。

# 第3章　アレイシミュレータの開発

アレイの状態をコンピュータ上で再現するためのプログラムを作成する。

## 3.1　使用言語

使用する言語にはPythonを採用する。Pythonはインタプリタ型の言語であり、コンパイルが必要ないためコードの修正・動作確認が行いやすくプロトタイピングに適している。また、Pythonの開発思想として単純で簡潔なコードで書けることを重視しているため、可読性が高く保守や引き継ぎのハードルが高くないことから採用した。

また、基本機能以外の専門的な機能や拡張モジュールがインターネット上に多く公開されており、簡単に導入できるため拡張性が高く、将来的に画像処理などの機能追加が必要となった場合にも対応ができるほど汎用なプログラミング言語である。

一方で、Pythonの処理速度は一般に遅いとされる水準で、コンパイラ型のC言語などと比較すると10〜100倍程度遅い。今後、機能を追加していく中で処理速度に問題が生じてきたらCython[*4]やC言語への移植も検討する。

## 3.2　構想

斜交座標系のグラフを作成し、ユニットの位置と搬送物の位置関係が視覚的にわかる図を出力するプログラムを作成する。

図3.1のように、任意の出発地点から目的地へのルートを計算し、通過するユニットの特定する機能を実装する。

[*4]Cython(サイソン) : Pythonの言語仕様を踏襲し、C言語の静的型付けなどを導入することによりPythonで律速となる配列処理やループ処理を高速化させたプログラミング言語。Pythonと比較して100倍以上高速となる場合がある。

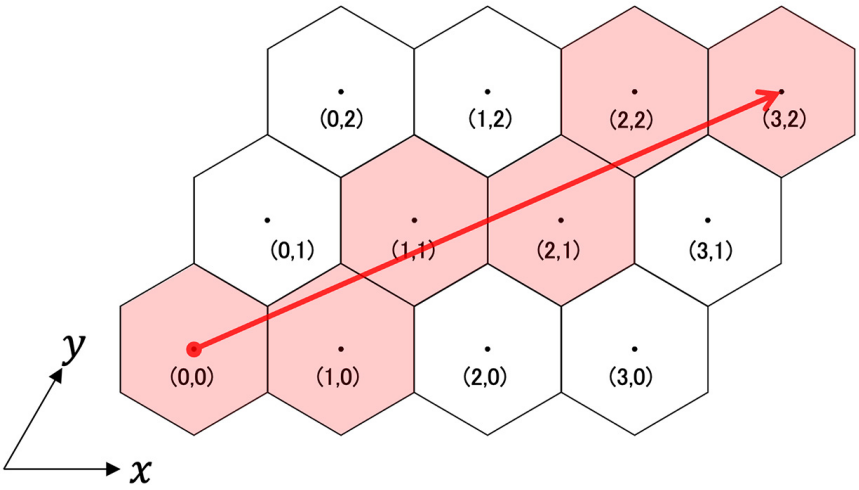



図 3.1 アレイシミュレータの構想

## 3.3 実装

### 3.3.1 斜交座標系の実装

グラフの描画には matplotlib というモジュールを使用する。Python がインストールされ、PATH が設定された状態でターミナル上に pip install matplotlib というコマンドを入力すると matplotlib がインストールされる。

斜交座標系の設定は matplotlib には実装されていないが、有志によるモジュールがインターネット上で公開されていたため、その一部分を利用した。[23] この斜交座標系を設定するモジュールを my_skewt.py とし、付録のソースコード A.1 に添付する。

my_skewt.py を使用した斜交座標系のプロットのテストの出力図を図 3.2 に示す。10×10 のアレイを表示し、原点 (0, 0) から (8, 5) に向かう直線を引いた。ユニットの描画には matplotlib で用意されている六角形のマーカを使用しているため、プログラム内では点としての情報しか持たないが、斜交座標系のグラフ表示は問題なくできていることが確認された。

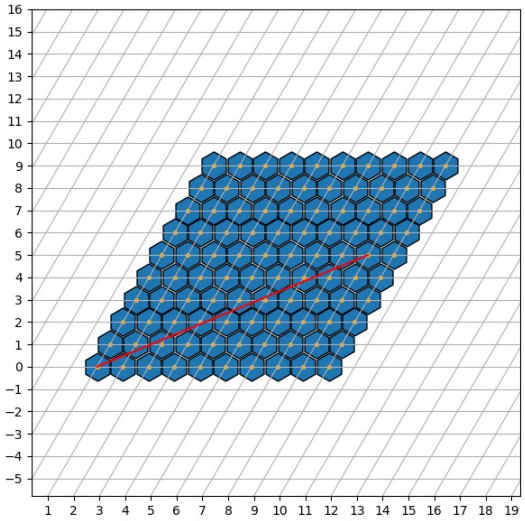



図 3.2 斜交座標系でのプロットテスト

### 3.3.2 ユニットの描画

斜交座標系のプロットが問題なく行えることがわかったため、次にユニットの描画方法の改善を行なった。隣り合うユニット間の距離を 1 とした時、外形の頂点座標は図 3.3(a) のようになる。これを式 2.6 を用いて座標変換すると図 3.3(b) が得られる。

得られた頂点座標をプログラムに導入し、プロットしたものを図 3.4 に示す。図の通り、隙間なくユニットの外形をプロットできていることがわかる。これによって各ユニットの領域をプログラムに読み込ませることができ、搬送物の座標がユニットの内にあるか外にあるかの内外判定に使用することができる。

内外判定は matplotlib.path モジュールの Path クラスに領域の頂点座標を渡し、contains_point メソッドを呼び出し搬送物の座標を引数に指定することで実装できる (ソースコード A.2 293 行目)。内外判定を実装した結果を図 3.5 に示す。図のように搬送物の直下のユニットのみが塗りつぶされ、内外判定が正しく行われていることがわかる。また、ユニットの境界線上に搬送物がある場合は隣り合う 2 つのユニット上に含まれるものとして判定される仕様であることがわかった (図 3.5(c))。

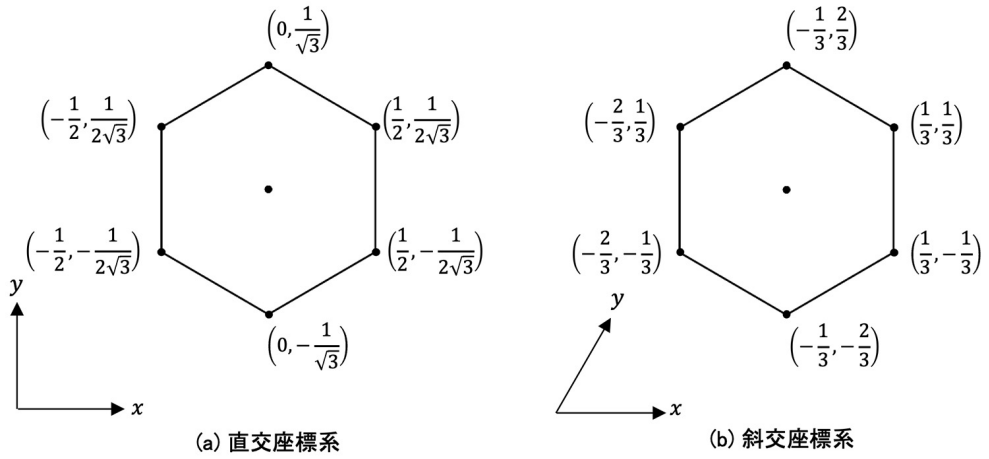



図 3.3 ユニット外形の頂点座標の座標変換

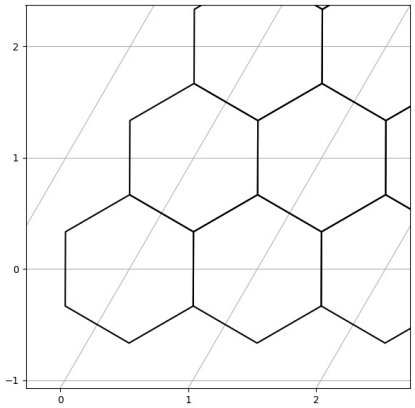


図 3.4 座標指定によるユニットの描画

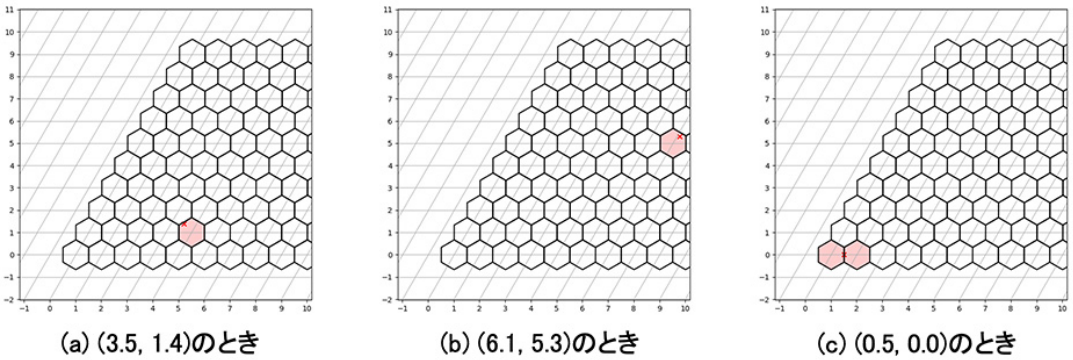



図 3.5 内外判定の結果

### 3.3.3　オブジェクト指向型プログラミング

また、今後の機能追加や修正を行いやすくするため、オブジェクト指向型にプログラムを修正した。ユニットやアレイの情報はクラス化し、main 関数内でオブジェクトとして呼び出すことで記述を簡略化できた (ソースコード A.2 269〜301 行目)。クラスの考え方を図 3.6 に示す。

ユニットに関するパラメータである座標やモータの駆動情報などを Unit クラスのインスタンス変数とし、ユニットの描画などの操作はクラス内のメソッドとして定義した。この Unit クラスを配列として継承したものが Array クラスである。

このような入れ子構造とすることで、例えば 3 つ目のユニットをグラフ上に描画する場合、main 関数内では Array.order[2].draw() で呼び出すことができ、コードを簡略化することができる。Python ではドット (.) はオブジェクトのメソッドや変数へアクセスする意味をもち、上の場合では Array というクラスの変数である order の 3 番目のオブジェクトにある draw() というメソッドを呼び出すという意味になる。[*5]

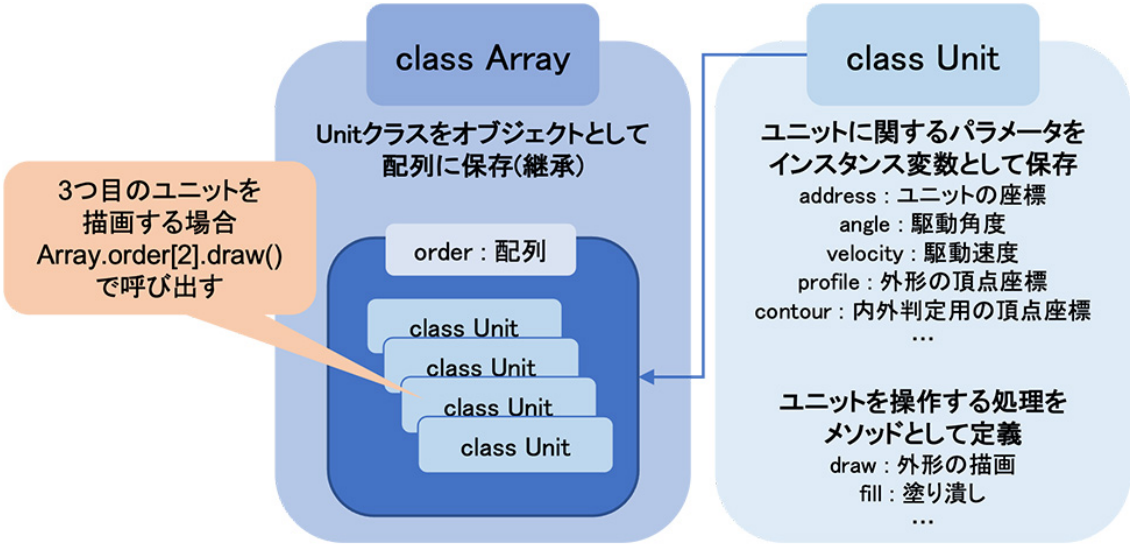



図 3.6 クラスの考え方

## 3.4　成果物

最終的に作成したプログラムの出力を図 3.7 に示す。構想段階での機能は実装でき、出発地点 (1, 2) から目的地 (4, 7) までの経路上で通過するユニットの特定ができた。出発地点と目的地の変更はソースコード A.2 の 274〜275 行目の start_x, start_y, destination_x, destination_y を変更することで可能である。

ここでは作動させるユニットの特定を行うだけで制御信号を出力するなどの機能は持たせていないが、このプログラムを作成することで得られたノウハウは後述する CAN 通信コントローラの開発で流用が可能である。

[*5]配列のインデックスは 0 から始まるため、3 番目の要素を表すインデックスは 2 となる。

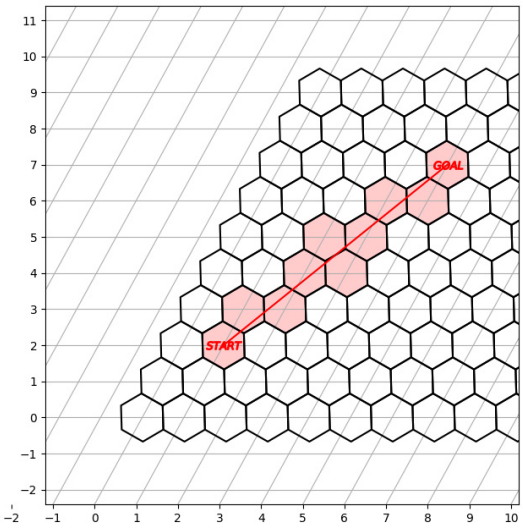



図 3.7 アレイシミュレータの出力画面

# 第4章　1対1シリアル通信コントローラの開発

## 4.1　先行研究

1.2.3 項で述べたように鈴鹿により Processing によるコントローラがすでに開発されている (図 1.15(b))。これにより速度と搬送角度を個別に制御できることは実証されていたが、モータごとにノブを調節して操作する方式であったため、2 つのモータを同時に制御することはできなかった (図 4.1)。実際に操作する際には個別に制御するより、飛行機などで用いられているジョイスティックのようなインターフェースを用いて 1 回の操作で速度と角度を制御できる方が直感的に操作ができ、操作性が向上すると考えられる。

また、サーボモータの制御が安定せず振動を繰り返す不具合があったため、ここではこれら 2 つの課題を解決し、より直感的に操作できるコントローラの開発を目指す。

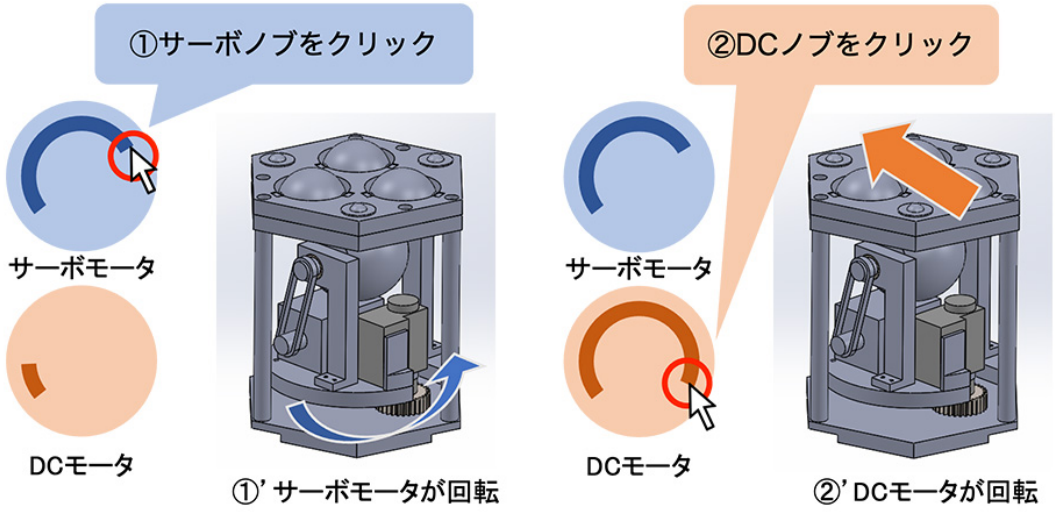



図 4.1 先行研究でのコントローラの動作図

## 4.2　構想

上述のようにジョイスティックのようなインターフェースで DC モータの駆動速度とサーボモータの角度を同時に制御するため、極座標グラフを活用することを考案した。極座標グラフ上の任意の点をクリックすることでクリックした点の座標を入力値として取得し、原点からの距離と偏角で DC モータの駆動速度とサーボモータの角度を制御する機能を実装する (図 4.2)。

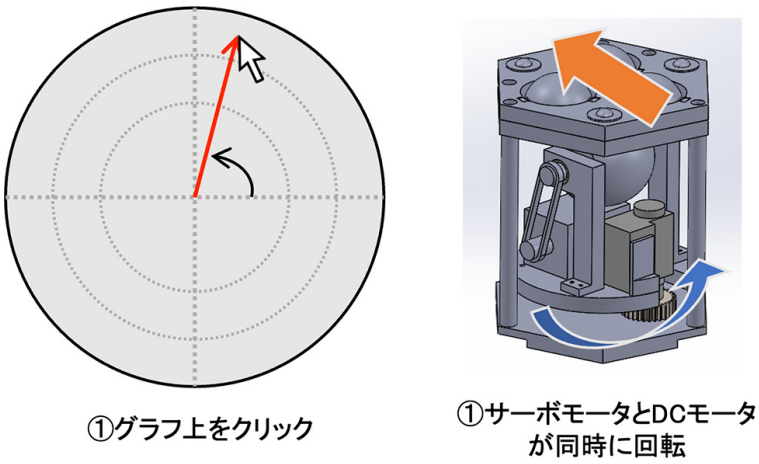



図 4.2 シリアルコントローラのコンセプト

## 4.3 Python の実装

### 4.3.1 極座標グラフの実装

matplotlib には極座標系のグラフを出力する polar projection が用意されているため、これを利用する。具体的な実装方法はインターネット上に公式ドキュメントが公開されていたため、それに従うことで難なく実装できた (ソースコード A.3 93〜97 行目)。[24]

### 4.3.2 クリック操作の機能実装

グラフ上でのクリック操作の処理については matplotlib.pyplot の figure.canvas.mpl_connect メソッドによって実装する (ソースコード A.3 113 行目)。“button_press_event” という記述はマウスのクリックを意味し、クリックされたとき onclick 関数が呼び出される。

onclick 関数ではクリック座標を取得し、原点からの距離を DC モータの駆動速度、x 軸との偏角をサーボモータの角度に設定し、制御信号を出力する処理を実装した (ソースコード A.3 26〜55 行目)。

制御信号は “VVVAAAVVVAAA;” の 13 桁の UTF-8 型で出力される。ここで “VVV” は 0 を左詰めし 3 桁にした DC モータの駆動速度、“AAA” は 0 を左詰めし三桁にしたサーボモータの角度である。また、“ ; ” は信号の終わりを示す終端文字である。例えば、DC モータの駆動速度を 255、サーボモータの角度を 90° とした場合、出力される制御信号は “255090255090;” となる。

また、“VVVAAA” を 2 回繰り返しているのは将来的に 2 つのユニットを個別に制御することを考慮したためである。

### 4.3.3　シリアル通信の実装

次に Arduino へ制御信号を出力するため、シリアル通信の機能を実装する。Python でのシリアル通信には pyserial というモジュールを使用する。これもターミナル上で pip install pyserial というコマンドを入力するとモジュールがインストールされる。

具体的な実装についてはインターネット上で公開されている公式ドキュメントを参考にして実装した。[25]

## 4.4　Arduino の実装

Arduino 側の実装については大部分は鈴鹿による先行研究で作成されたものを流用したが、動作確認で以前から問題となっていたサーボモータの振動が確認されたため、原因究明のためのデバッグを行なった。デバッグの結果、シリアルデータの受信処理が適切ではなく終端文字の検出ができず、意図していない文字列がサーボモータの制御値に入力されていたことがわかった。

for ループを用いて Serial.read 関数で 1 文字ずつ受信し、char 型の配列に代入していたことが原因であると考えられたため、この処理を Serial.readStringUntil 関数に変更し終端文字 “ ; ” が検出されるまでのシリアルデータを文字列として一括で受信する方式に変更した。


図 4.3 serial_controller.ino の変更点

## 4.5　成果物

作成したプログラムの出力画面を図 4.4 に示す。極座標のグラフが表示され、グラフ上をクリックすることで選択点までの距離と角度を取得し、DC モータとサーボモータの同時制御が行われる。4.4 節での修正により、サーボモータの振動問題も解決することができた。

このプログラムはイノベーション・ジャパン 2019[*6]でユークリーターの展示を行った際にも使用され、動作実証に役立った。

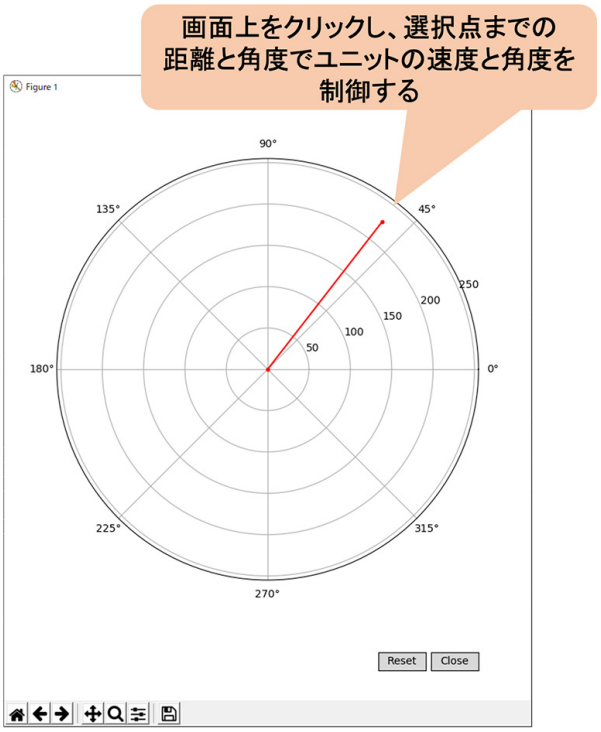



図 4.4 シリアルコントローラの出力画面

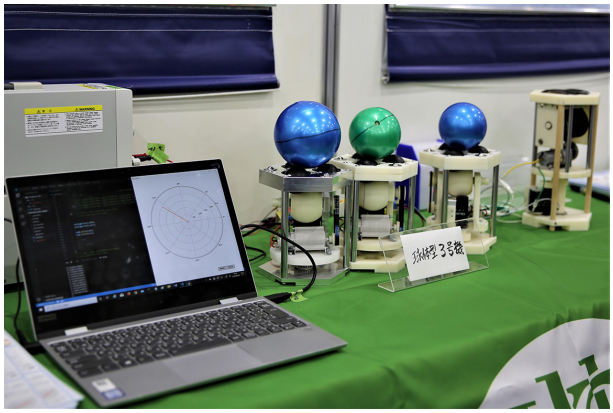


図 4.5 イノベーション・ジャパン 2019 での展示の様子

[*6]東京ビッグサイト青海展示棟 B ホール, 東京, 日本, 2019 年 8 月 29-30 日. 小間 : S-15, Pre : 8 月 30 日 11:20-11:30

# 第5章　1対多数CAN通信コントローラの開発

## 5.1　先行研究

先行研究において鈴鹿により CAN 通信を用いた Processing によるコントローラがすでに開発されている (図 1.17(a))。このコントローラではトグルスイッチにより送信先の ID を選択し、スライダにより駆動速度と角度を制御するプログラムとなっている。本コントローラを用いて複数個のユニットを個別に制御できることが実証されていた。しかしながらトグルスイッチを同時に操作することができない仕様であり、たとえできるようにしたとしても操作性に限界があることは明らかである。

ここではより使い勝手が良く、連携して搬送物の搬送が行えるようなプログラムを作成する。

## 5.2　構想

第 3 章で作成したプログラムをベースとしてアレイを表示し、グラフ上をクリックで目的地を設定、マウスカーソル位置を搬送物の現在地として取得する機能を実装し、内外判定により現在地に対応したユニットを駆動させるプログラムとする。

Python から Arduino(CAN ジャンクション基板) へのシリアル通信は第 4 章で作成したプログラムを流用する。シリアル通信で送信するデータは “XXXXYYYYDVVVAAA;” の 16 桁とする。ここで “XXXXYYYY” はユニットの ID(8bit の 2 進数)、“D” は駆動方向 (1 : 正転, 2 : 逆転)、“VVV” は D モータの駆動速度、“AAA” はサーボモータ角度、“ ; ” は終端文字とする。

また、CAN ジャンクション基板からドライバ基板への CAN 通信の処理は先行研究で作成されたものをベースに実装する。

## 5.3　実装

### 5.3.1　アレイ情報の csv 入力の実装

第 3 章で作成したプログラムでは簡単のため、アレイは ($x$ 個 $\times$ $y$ 個) の長方形のみを出力するようにしていたが、現実には曲がった空間や柱などの障害物を避ける形にユニットを敷き詰めることが考えられるため、自由な形のアレイを作成できるように csv での入力に対応するよう機能を追加した。

アレイの情報は array_config.csv に表 5.1 のように記述する。これを新たに作成した FlexArray クラスに引数として渡すことで任意のアレイをプログラムに読み込ませる機能を実装した (ソースコード A.6 356〜358 行目)。図 5.1 に FlexArray クラスの概念図を示す。

表 5.1 array_config.csv の入力例

| ID | position_x | position_y |
|---|---|---|
| 00010001 | 1 | 1 |
| 00100001 | 2 | 1 |
| 00010010 | 1 | 2 |
| 00100010 | 2 | 2 |
| ⋮ | ⋮ | ⋮ |

csv の読み込みには pandas モジュールを使用する。ターミナル上で pip install pandas というコマンドを入力することで pandas がインストールされる。実装方法についてはインターネット上で公開されている公式ドキュメントを参考にした。[26]

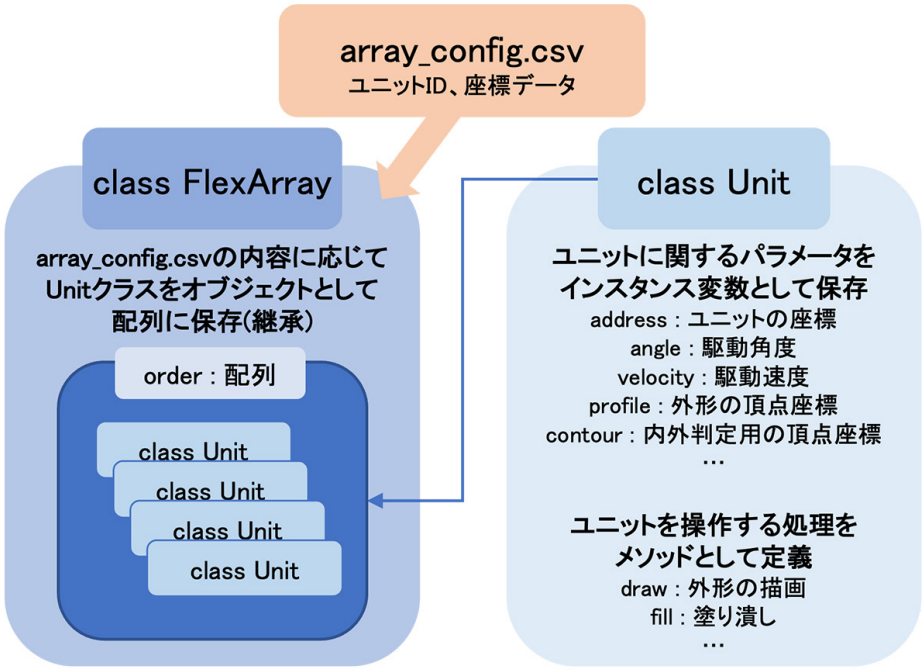



図 5.1 FlexArray クラスの概念図

### 5.3.2 マウス操作の機能実装

グラフ上でのマウス操作の処理については 4.3.2 項と同様に figure.canvas.mpl_connect メソッドによって実装できる。第 1 引数を “motion_notify_event” とすることでマウスカーソルが移動した際に motion 関数が呼び出される (ソースコード A.6 371 行目)。

motion 関数ではマウスカーソルの位置を取得し、各ユニットとの内外判定を行う。内外判定によりいずれかのユニット内にマウスカーソルがあるとき、クリック操作により設定された目的地へのベクトルを計算し、偏角を計算する。得られた偏角を Unit クラスの set_on_route 関数に渡す

ことでそのユニットの元の角度から変化があったかの判定が行われ、変化があった場合 Unit クラスのインスタンス変数である self.on_changed が True となる。この判定で True となった場合、メッセージをシリアル通信で出力する (ソースコード A.6 308〜343 行目)。

これらの処理により、ユニットの駆動パラメータに変更があった時のみシリアル出力され、通信帯域を節約することができる (図 5.2)。

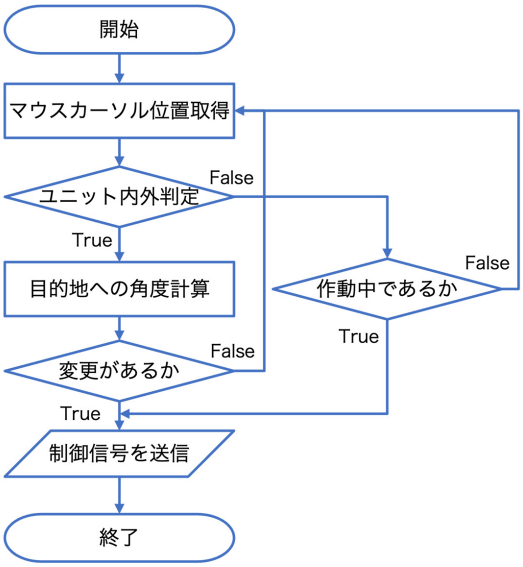



図 5.2 マウス操作時のフローチャート

## 5.4 成果物

作成したプログラムによる動作試験の様子を図 5.3 に示す。図右側の操作画面では (3, 1) のユニット上にマウスカーソルがあり、目的地に向かい左方向にユニットが駆動する状態になっている。この時、敷き詰めたユニットも意図通りに作動し、目的地の方向へ駆動した。マウスカーソルを左方向に移動させていくと、順番に作動するユニットが切り替わり連動していることが確認された。

また、操作画面でユニット上からマウスカーソルが外に出たユニットは駆動を止め、サーボモータが 90° となる原点回帰も正常に動作した。

駆動方向による DC モータの正転・逆転も正常に動作することが確認でき、実装した機能が全て問題なく動作することが確認された。

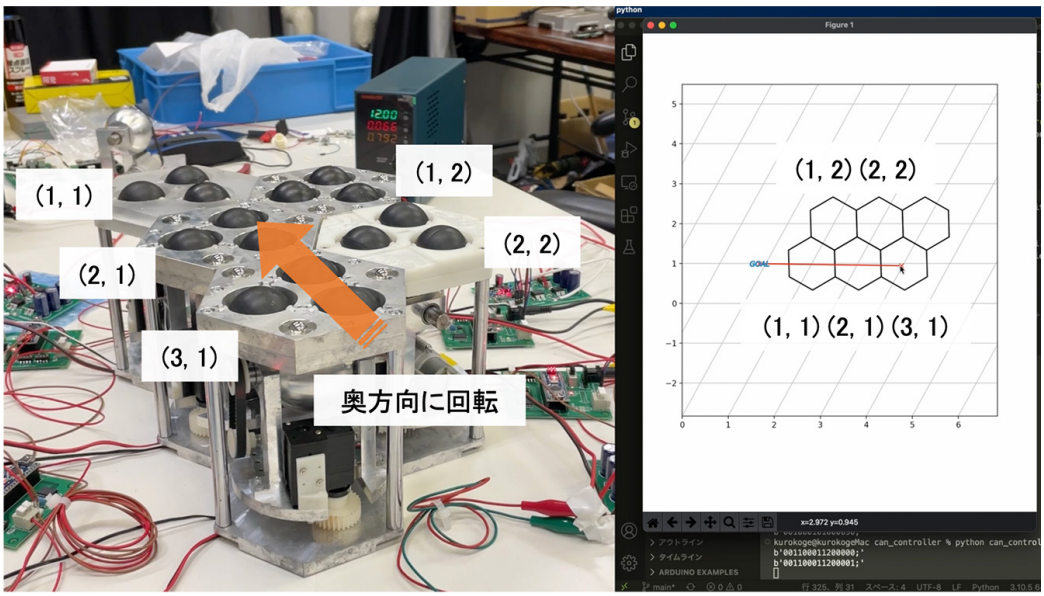

(1, 1)
(1, 2)
(2, 2)
(2, 1)
(3, 1)
奥方向に回転
python
Figure 1
(1, 2) (2, 2)
GOAL
(1, 1) (2, 1) (3, 1)
x=2.972 y=0.945
kurokoge@kurokogeMac can_controller % python can_control
b'00110001120000;'
b'00110001120001;'
アウトライン
タイムライン
ARDUINO EXAMPLES

図 5.3 動作試験の様子

# 第6章　結論

## 6.1　要約

本研究は次世代の移動手段「ユークリーター」をアレイ化した際の制御システムの開発に関する研究である。本研究では先行研究をもとに実用化に向けた統合制御のソフトウェア開発を行った。以下に各章の総括を述べる。

第 1 章では人類の移動手段の歴史について述べ、現代の移動手段における問題点を提起した。そこで次世代の移動手段として我々の研究室で開発を行っているユークリーターを提案した。ユークリーターが実現するとこれまであまり開発されてこなかった近場や密集地での移動をより安全に、より高効率に行えることを述べた。
次にユークリーターの機構系および制御系の先行研究について説明をし、類似の研究との比較、新規性について述べ、本研究の目的について述べた。

第 2 章ではユークリーターをアレイ化する際に考えられる 2 つの検討課題について述べた。アレイ化したユークリーターへのアドレスの割り振りについて、直交座標系では不都合があり 60° 斜交座標系を用いることで解決できることを示した。また、膨大な数になる制御配線をシンプルかつ拡張性や保守性に優れたものにするため通信方式について検討し、シリアル通信や EtherCAT、CAN 通信についてそれぞれの特徴やメリット/デメリットについて述べ、CAN 通信をユークリーターの制御系に採用することを述べた。

第 3 章ではコンピュータ上でアレイの状態を再現するプログラムの実装について述べ、Python を用いて斜交座標系のグラフを作成し、アレイを再現しユニットの内外判定を実装することで出発地点から目的地への経路上で通過するユニットの特定ができた。

第 4 章ではシリアル通信を用いた 1 対 1 でのコントローラの開発について述べた。Python matplotlib モジュールの極座標グラフを活用し、グラフィカルで直感的に操作しやすいコントローラを作成した。また、Arduino のプログラムを見直し、先行研究で問題となっていたサーボモータの振動を解決することができ、安定した制御が可能となった。

第 5 章では CAN 通信を用いた 1 対多数のコントローラの開発について述べた。第 3 章、第 4 章の開発で得られたノウハウを集結し、アレイ化したユークリーターを統合制御するプログラムの作成を行った。動作試験において良好な結果が得られ、ユニットを連携させて作動させることで任意の場所へ搬送物を搬送するシステムの基盤が構築された。

以上により、ユークリーターの制御系は物の搬送ができる状態まで推進し、実用化に向けた大きな一歩を踏み出せたと考える。

## 6.2 展望

現在の CAN 通信を用いたコントローラでは搬送物の大きさは考慮されず、点として扱っているため同時に作動するユニットは 1 つだけとなる。現実には搬送物を大きさに応じて複数のユニットが作動しなければならないため、まずは搬送物の大きさをプログラムに実装することが第一目標である。

また、搬送物が複数ある場合や障害物があっても衝突しない制御方法の実装も必要である。そのためにはカメラやセンサなど何らかの検知器を用いたフィードバックシステムが必要であり、これらの実装も早期に実現したい。

そのほか、DC モータへのエンコーダを設置し、速度や加速度の制御も行う必要があると考える。ユークリーターは最終的には人の移動を行うことを目標としているため、乗り心地の良い加速・減速を行えるような機構を開発する必要がある。

# 参考文献


[1] 田辺あらし. 乗り物の歴史 - Science Portal. https://scienceportal.jst.go.jp/gateway/sciencewindow/20180731_w01/, (参照 2022/09/09).

[2] 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, 2017, 修士論文.

[3] 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, 2016, 卒業論文.

[4] 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, 2017, 卒業論文.

[5] 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, 2017, 卒業論文.

[6] 竹中克昭. ホイール配置による球体の全方向回転制御機構の開発. 高知工科大学, 2017, 卒業論文.

[7] 狩野大輝. 球体伝達機構を用いた全方向移動手段の開発. 高知工科大学, 2018, 卒業論文.

[8] 鈴鹿紅音. 球体と全方向移動制御装置を用いた次世代移動手段の開発. 高知工科大学, 2018, 卒業論文.

[9] 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, 2019, 卒業論文.

[10] 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, 2020, 修士論文.

[11] 竹中克昭. 次世代型全方向輸送機構の開発. 高知工科大学, 2019, 修士論文.

[12] 石井和磨. ベルトを用いた全方向移動装置の開発. 高知工科大学, 2019, 卒業論文.

[13] 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, 2021, 修士論文.

[14] 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, 2020, 修士論文.

[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, 2021, 卒業論文.

[16] 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, 2021, 卒業論文.

[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, 2022, 卒業論文.

[18] cellumation GmbH. The celluveyor - The innovation through cells. https://cellumation.com/products/celluveyor/, (参照 2022/08/07).

[19] Beckhoff Automation GmbH. XPlanar - Planar motor system. https://www.beckhoff.com/en-us/products/motion/xplanar-planar-motor-system/, (参照 2022/08/07).

[20] 石田秀一, 宮本弘之. 球体駆動式全方向移動機構の開発. 日本機械学会論文集 C 編. 2012, Vol. 78, No. 790, p. 227-235.

[21] 安藤恒也. グラフェンの特異な物理. 表面科学. 2008, Vol. 29, No. 5, p. 296-303.

[22] 菱山幸宥, 鏑木 裕. 単層グラフェンにおける強結合ハミルトニアンとエネルギー分散. 炭素. 2019, Vol. 2019, No. 289, p. 159-171.

[23] 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).

[24] The Matplotlib development team. Polar plot - Matplotlib 3.5.3 documentation. https://matplotlib.org/stable/gallery/pie_and_polar_charts/polar_demo.html, (参照 2022/08/17).

[25] Chris Liechti. pySerial API - pySerial 3.0 documentation. https://pythonhosted.org/pyserial/pyserial_api.html, (参照 2022/08/17).

[26] the pandas development team. User Guide - pandas 1.4.3 documentation. https://pandas.pydata.org/docs/user_guide/index.html, (参照 2022/08/17).

# 謝辞

本研究は高知工科大学大学院 工学研究科 基盤工学専攻 知能機械システム工学コース 材料革新サスティナブルテクノロジー研究室において、川原村敏幸教授のご指導の下行われました。ユークリーターに関係する研究が行いたいという私の希望を汲んでくださり、本研究のテーマであるユークリーターの制御システムの開発という重要かつやりがいのあるテーマを与えてくださいました。

本研究の遂行にあたり、プログラミングやマイコンの知識など、川原村敏幸教授には専門分野ではないものの深く理解していただき、多くの気づきやアイデアをいただきました。また、体調が優れず学業や研究にうまく取り組めない時期があり、ご迷惑をお掛けしたこともあったかと思いますが、そんな中でもいつも気にかけてくださり焦らず自分のペースで進められるようにお心遣いを賜りました。苦難の多い道のりではありましたが、最終的に研究成果を本論文にまとめ上げることができたことは川原村敏幸教授のご指導無くしてなし得ませんでした。ここに深謝の意を表します。

教育講師室の重森総一郎先生、健康相談室の皆様には研究のみならず学生生活での悩みなど公私に渡り様々な相談に応じていただき、サポートしていただきました。親身になって相談に乗っていただける存在があったことは学生生活を送る上で心強い支えになりました。厚く御礼申し上げます。

共に同じ分野の研究を行なってきた、狩野大輝さん、鈴鹿紅音さん、石井和磨さん、松坂康永さん、亀岡正樹さん、高井友輝さん、澤田陸斗さんとはユークリーターの機構や制御について議論を重ね、よりよいものを作り上げるため互いに高め合ってきました。また、実験を行う際にもご協力をお願いし快くお引き受けいただきました。心より感謝いたします。

研究室の同学年である、石井和磨さん、尾崎珠子さん、安岡龍哉さんには大学院進学時に研究室を移ってきた私を温かく迎え入れてくださり、食事に誘っていただいたり親睦を深める機会を設けていただきました。おかげさまで引け目を感じることなく研究室での生活に馴染むことができました。重ねて感謝申し上げます。

また、研究室の皆様には研究会や発表練習において様々な意見、質問をしていただきました。物事を多面的に見ることで考えが独りよがりにならず、資料をよりよいものにブラッシュアップでき、自分の成長にも繋がったと思います。深く感謝いたします。

最後になりましたが、人生最高の贅沢とも言える大学院での学問を何不自由なく続けられるようご支援いただき、温かく見守ってくださいました両親、祖母に謹んで御礼申し上げます。これからも工学を究め続け、社会に良い影響を与えられるような存在になれるよう精進する所存です。

# 付 録A　作成したソースコード

作成したソースコードを付録として添付する。

## A.1　第3章で作成したソースコード

アレイシミュレータの開発で作成したプログラム。

array_route_check.py が保存されているディレクトリに my_module フォルダを作成し、その中に my_skwet.py を保存する。

#### 動作環境

- Python v3.10.5
- numpy v1.23.1
- matplotlib v3.5.2

### A.1.1　Python

ソースコード A.1 my_skewt.py

```
"""
============================================================
SkewT-logP diagram: using transforms and custom projections
============================================================

This serves as an intensive exercise of matplotlib's transforms and custom
projection API. This example produces a so-called SkewT-logP diagram, which is
a common plot in meteorology for displaying vertical profiles of temperature.
As far as matplotlib is concerned, the complexity comes from having X and Y
axes that are not orthogonal. This is handled by including a skew component to
the basic Axes transforms. Additional complexity comes in handling the fact
that the upper and lower X-axes have different data ranges, which necessitates
a bunch of custom classes for ticks,spines, and the axis to handle this.

"""

from matplotlib.axes import Axes
import matplotlib.transforms as transforms
import matplotlib.axis as maxis
import matplotlib.spines as mspines
from matplotlib.projections import register_projection


# The sole purpose of this class is to look at the upper, lower, or total
# interval as appropriate and see what parts of the tick to draw, if any.
```

```
class SkewXTick(maxis.XTick):
    def update_position(self, loc):
        # This ensures that the new value of the location is set before
        # any other updates take place
        self._loc = loc
        super(SkewXTick, self).update_position(loc)

    def _has_default_loc(self):
        return self.get_loc() is None

    def _need_lower(self):
        return (self._has_default_loc() or
                transforms.interval_contains(self.axes.lower_xlim,
                                             self.get_loc()))

    def _need_upper(self):
        return (self._has_default_loc() or
                transforms.interval_contains(self.axes.upper_xlim,
                                             self.get_loc()))

    @property
    def gridOn(self):
        return (self._gridOn and (self._has_default_loc() or
                transforms.interval_contains(self.get_view_interval(),
                                             self.get_loc())))

    @gridOn.setter
    def gridOn(self, value):
        self._gridOn = value

    @property
    def tick1On(self):
        return self._tick1On and self._need_lower()

    @tick1On.setter
    def tick1On(self, value):
        self._tick1On = value

    @property
    def label1On(self):
        return self._label1On and self._need_lower()

    @label1On.setter
    def label1On(self, value):
        self._label1On = value

    @property
    def tick2On(self):
        return self._tick2On and self._need_upper()

    @tick2On.setter
    def tick2On(self, value):
        self._tick2On = value

    @property
    def label2On(self):
        return self._label2On and self._need_upper()

    @label2On.setter
    def label2On(self, value):
        self._label2On = value

    def get_view_interval(self):
        return self.axes.xaxis.get_view_interval()
```

```
90
91
92  # This class exists to provide two separate sets of intervals to the tick,
93  # as well as create instances of the custom tick
94  class SkewXAxis(maxis.XAxis):
95      def _get_tick(self, major):
96          return SkewXTick(self.axes, None, major=major)
97
98      def get_view_interval(self):
99          return self.axes.upper_xlim[0], self.axes.lower_xlim[1]
100
101
102 # This class exists to calculate the separate data range of the
103 # upper X-axis and draw the spine there. It also provides this range
104 # to the X-axis artist for ticking and gridlines
105 class SkewSpine(mspines.Spine):
106     def _adjust_location(self):
107         pts = self._path.vertices
108         if self.spine_type == 'top':
109             pts[:, 0] = self.axes.upper_xlim
110         else:
111             pts[:, 0] = self.axes.lower_xlim
112
113
114 # This class handles registration of the skew-xaxes as a projection as well
115 # as setting up the appropriate transformations. It also overrides standard
116 # spines and axes instances as appropriate.
117 class SkewXAxes(Axes):
118     # The projection must specify a name.  This will be used be the
119     # user to select the projection, i.e. ``subplot(111,
120     # projection='skewx')``.
121     name = 'skewx'
122
123     def _init_axis(self):
124         # Taken from Axes and modified to use our modified X-axis
125         self.xaxis = SkewXAxis(self)
126         self.spines['top'].register_axis(self.xaxis)
127         self.spines['bottom'].register_axis(self.xaxis)
128         self.yaxis = maxis.YAxis(self)
129         self.spines['left'].register_axis(self.yaxis)
130         self.spines['right'].register_axis(self.yaxis)
131
132     def _gen_axes_spines(self):
133         spines = {'top': SkewSpine.linear_spine(self, 'top'),
134                   'bottom': mspines.Spine.linear_spine(self, 'bottom'),
135                   'left': mspines.Spine.linear_spine(self, 'left'),
136                   'right': mspines.Spine.linear_spine(self, 'right')}
137         return spines
138
139     def _set_lim_and_transforms(self):
140         """
141         This is called once when the plot is created to set up all the
142         transforms for the data, text and grids.
143         """
144         rot = 30
145
146         # Get the standard transform setup from the Axes base class
147         Axes._set_lim_and_transforms(self)
148
149         # Need to put the skew in the middle, after the scale and limits,
150         # but before the transAxes. This way, the skew is done in Axes
151         # coordinates thus performing the transform around the proper origin
152         # We keep the pre-transAxes transform around for other users, like the
153         # spines for finding bounds
```

```
        self.transDataToAxes = self.transScale + \
            self.transLimits + transforms.Affine2D().skew_deg(rot, 0)

        # Create the full transform from Data to Pixels
        self.transData = self.transDataToAxes + self.transAxes

        # Blended transforms like this need to have the skewing applied using
        # both axes, in axes coords like before.
        self._xaxis_transform = (transforms.blended_transform_factory(
            self.transScale + self.transLimits,
            transforms.IdentityTransform()) +
            transforms.Affine2D().skew_deg(rot, 0)) + self.transAxes

    @property
    def lower_xlim(self):
        return self.axes.viewLim.intervalx

    @property
    def upper_xlim(self):
        pts = [[0., 1.], [1., 1.]]
        return self.transDataToAxes.inverted().transform(pts)[:, 0]


# Now register the projection with matplotlib so the user can select
# it.
register_projection(SkewXAxes)
```

ソースコード A.2 array_route_check.py

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.path as mtplpath

from my_module import my_skewt

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
    ----------
    address_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
    on_route : bool
        内外判定の結果
    """
    def __init__(self, address_x: int, address_y: int):
        """initialize.

        Parameters
```

```
35          ----------
36          address_x : int
37              ユニットのx座標
38          address_y : int
39              ユニットのy座標
40          """
41          self.address_x = address_x
42          self.address_y = address_y
43          self.angle = 0.0
44          self.velocity = 0.0
45
46          # 斜交座標系での六角形座標への補正量
47          hex_profile_x = np.array([1.0/3.0, -1.0/3.0, -2.0/3.0, -1.0/3.0, 1.0/3.0, 2.0/3.0,
                1.0/3.0])
48          hex_profile_y = np.array([1.0/3.0, 2.0/3.0, 1.0/3.0, -1.0/3.0, -2.0/3.0, -1.0/3.0,
                1.0/3.0])
49
50          # 中心点の調整
51          self.profile_x = hex_profile_x + float(address_x)
52          self.profile_y = hex_profile_y + float(address_y)
53
54          # 内外判定用
55          self.contour = []
56          for i in range(7):
57              self.contour.append([self.profile_x[i], self.profile_y[i]])
58
59          # 経路上にあるかどうかの判定用
60          self.on_route = False
61
62      def draw(self) -> None:
63          """外形をグラフに描画する関数."""
64          plt.plot(self.profile_x, self.profile_y, color='black')
65          # plt.plot(self.address_x, self.address_y, color='black', marker='.')
66          return None
67
68      def fill(self) -> None:
69          """ユニットを塗りつぶす関数."""
70          plt.fill(self.profile_x, self.profile_y, facecolor='red', alpha=0.2)
71          return None
72
73      def set_angle(self, angle: int) -> None:
74          """ユニットの角度を設定する関数."""
75          self.angle = angle
76          return None
77
78      def on(self, velocity: int) -> None:
79          """ユニットの速度を設定する関数."""
80          self.velocity = velocity
81          return None
82
83  class Array:
84      """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.
85
86      Attributes
87      ----------
88      address_x : int
89          アレイ内のユニットのx座標
90      address_y : int
91          アレイ内のユニットのy座標
92      order : list
93          アレイ内のユニットクラスを継承
94
95      See Also
96      --------
```

```
    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, row: int, column: int):
        """initialize.

        Parameters
        ----------
        row : int
            アレイの行数
        column : int
            アレイの列数
        """
        size = (row, column)
        self.address_x = np.empty(size)
        self.address_y = np.empty(size)
        self.order = []

        for y in range(column):
            for x in range(row):
                self.address_x[x][y] = x
                self.address_y[x][y] = y
                self.order.append(Unit(x, y))

class Object:
    """輸送対象物の各種パラメータを格納しているクラス.

    Attributes
    ----------
    position_x : float
        搬送対象物のx座標
    position_y : float
        搬送対象物のy座標
    size : float
        搬送対象物の大きさ
    shape : str
        搬送対象物の外形
    shape_fineness : int
        搬送対象物の外形の解像度
    shape_contour_x : list of float
        搬送対象物の外形の輪郭のx座標
    shape_contour_y : list of float
        搬送対象物の外形の輪郭のy座標
    destination_x : float
        目的地のx座標
    destination_y : float
        目的地のy座標
    path_x : list of float
        現在地から目的地への経路のx座標
    path_y : list of float
        現在地から目的地への経路のy座標
    """
    def __init__(self, start_x: float, start_y: float, size: float, shape: str, fineness : int
        ):
        """initialize.

        Parameters
        ----------
        start_x : float
            搬送対象物の初期位置のx座標
        start_y : float
            搬送対象物の初期位置のy座標
        size : float
```

```
160             搬送対象物の大きさ
161         shape : str
162             搬送対象物の形状 (p : 点, c : 円形)
163         fineness : int
164             搬送対象物の形状の解像度
165         """
166         self.position_x = start_x
167         self.position_y = start_y
168         self.size = size
169         self.shape = shape
170         self.shape_fineness = fineness
171         if self.shape == "c":
172             self.shape_contour_x = np.zeros(self.shape_fineness + 1)
173             self.shape_contour_y = np.zeros(self.shape_fineness + 1)
174             for i in range(self.shape_fineness):
175                 self.shape_contour_x[i] = self.position_x +0.5*self.size*np.cos(i/self.
                        shape_fineness*2*np.pi)
176                 self.shape_contour_y[i] = self.position_y + 0.5*self.size*np.sin(i/self.
                        shape_fineness*2*np.pi)
177             self.shape_contour_x[self.shape_fineness] = self.shape_contour_x[0]
178             self.shape_contour_y[self.shape_fineness] = self.shape_contour_y[0]
179         elif self.shape == "p":
180             self.shape_fineness = 1
181             self.shape_contour_x = [self.position_x]
182             self.shape_contour_y = [self.position_y]
183         else:
184             print("Type Error!")
185             exit()
186 
187     def move(self, velocity: float, angle: float, delta: float) -> None:
188         """搬送対象物を移動させる関数.
189 
190         self.position_x, self.position_yが内部で更新される
191 
192         Parameters
193         ----------
194         velocity : float
195             速度(スカラー量)[m/s]
196         angle : float
197             角度(度数法)[deg]
198         delta : float
199             微小時間[s]
200         """
201         self.position_x = self.position_x + velocity*np.cos(angle*np.pi/180)*delta
202         self.position_y = self.position_y + velocity*np.sin(angle*np.pi/180)*delta
203         return None
204 
205     def set_route(self, destination_x: float, destination_y: float) -> None:
206         """搬送対象物の現在地から目的地までの直線ルートを設定する関数.
207 
208         Parameters
209         ----------
210         destination_x : float
211             目的地のx座標
212         destination_y : float
213             目的地のy座標
214         """
215         self.destination_x = destination_x
216         self.destination_y = destination_y
217         # self.path_x = (self.position_x, destination_x)
218         # self.path_y = (self.position_y, destination_y)
219         self.path_x = np.linspace(self.position_x, destination_x, 20)
220         self.path_y = np.linspace(self.position_y, self.destination_y, 20)
221         return None
```

```
222
223 def skewt_plt_setup() -> tuple:
224     """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.
225
226     Returns
227     -------
228     fig : Figure
229     ax : Axes
230     """
231     fig = plt.figure(figsize=(7, 8.5))
232     # ax = fig.add_subplot(111, projection='skewx') # old
233     ax = fig.add_axes([0.1, 0.2, 0.8, 0.693], projection='skewx')
234     # 'equal'では高さが合わない -> √3/2倍する
235     ax.set_aspect(np.sqrt(3)/2, 'datalim')
236     ax.grid(True)
237     ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
238     ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
239     return fig, ax
240
241 def calc2ctrl(angle: float) -> float:
242     """計算上の角度を制御用の角度に変換する.
243
244     Parameters
245     ----------
246     angle : float
247         角度(弧度法)[rad]
248
249     Returns
250     -------
251     converted_angle : float
252         変換後の角度[rad](0<rad<2pi)
253     """
254     while angle >= 2*np.pi:
255         angle = angle - 2*np.pi
256     if 0 < angle and angle <= 90*np.pi/180:
257         converted_angle = angle*60/90
258     elif 90*np.pi/180 < angle and angle <= 180*np.pi/180:
259         converted_angle = 60*np.pi/180 + (angle - 90*np.pi/180)*120/90
260     elif 180*np.pi/180 < angle and angle <= 270*np.pi/180:
261         converted_angle = 180*np.pi/180 + (angle - 180*np.pi/180)*60/90
262     elif 270*np.pi/180 < angle and angle <= 360*np.pi/180:
263         converted_angle = 240*np.pi/180 + (angle - 270*np.pi/180)*120/90
264     elif angle == 0:
265         converted_angle = angle
266     return converted_angle
267
268
269 if __name__ == '__main__':
270     # アレイの設定
271     array = Array(row=10, column=10)
272
273     # 搬送対象物の設定
274     obj = Object(start_x=1, start_y=2, size=0.8, shape="p", fineness=20)
275     obj.set_route(destination_x=4, destination_y=7)
276
277     # グラフのセットアップ、ユニットの描画
278     fig, ax = skewt_plt_setup()
279     for unit in array.order:
280         unit_shape = unit.draw()
281
282     # 搬送対象物の現在地と目的地を描画
283     plt.plot(obj.position_x, obj.position_y, marker='$START$', color='red', markersize=30)
284     plt.plot(obj.destination_x, obj.destination_y, marker="$GOAL$", color="red", markersize
        =25)
```

```
285
286     # 搬送対象物の外形のプロット
287     plt.plot(obj.shape_contour_x, obj.shape_contour_y)
288     plt.plot(obj.path_x, obj.path_y, color='red', marker='')
289
290     # 内外判定
291     for i in range(len(obj.path_x)):
292         for unit in array.order:
293             if mtplpath.Path(unit.contour).contains_point([obj.path_x[i], obj.path_y[i]]) ==
                    True:
294                 unit.on_route=True
295
296     # 搬送対象物直下のユニットを塗りつぶす
297     for unit in array.order:
298         if unit.on_route == True:
299             unit.fill()
300
301     plt.show()
```

## A.2　第4章で作成したソースコード

シリアル通信を用いた1対1のコントローラの開発で作成したプログラム。

PC(Python)で制御パラメータを決定し、モータドライバ基板(Arduino)へシリアル通信で制御信号を送信する。

#### 動作環境

- Python v3.10.5
- numpy v1.23.1
- matplotlib v3.5.2
- pyserial v3.5

### A.2.1　Python

ソースコード A.3 serial_controller.py

```
#!/usr/bin/env python3
import sys
import time

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button
import serial

def reset(mouse_event) -> None:
    """Resetボタンを押したときに呼び出す関数."""
    point.set_data([0, 0], [0, 0])
    data = bytes('000000000000;', 'utf-8')
    ser.write(data)
    # ser2.write(data)
    print(data)
    return None

def close(mouse_event) -> None:
    """Closeボタンを押したときに呼び出す関数."""
    ser.close()
    # ser2.close()
    plt.close()
    sys.exit()

def onclick(event) -> None:
    """グラフ上をクリックしたときに呼び出す関数.

    Notes
    -----
    マウスカーソルの位置からユニットの駆動速度、角度を決定する
    制御パラメータの桁数に応じて0を左づめしてメッセージを送信する
    """
    if int(event.y) > 160:
        point.set_data([0, event.xdata], [0, event.ydata])
        fig.canvas.draw_idle()
        v = str(int(event.ydata))
        a = str(int(np.rad2deg(event.xdata)))
```

```
39
40          if len(v) == 3:
41              msg_v = str(v)
42          elif len(v) == 2:
43              msg_v = '0' + str(v)
44          elif len(v) == 1:
45              msg_v = '00' + str(v)
46
47          if len(a) == 3:
48              msg_a = str(a)
49          elif len(a) == 2:
50              msg_a = '0' + str(a)
51          elif len(a) == 1:
52              msg_a = '00' + str(a)
53
54          send(msg_v, msg_a)
55      return None
56
57  def send(velocity: int, angle: int) -> None:
58      """シリアルポートにメッセージを送信する関数.
59
60      Parameters
61      ----------
62      velocity : int
63          ユニットの駆動速度
64      angle : int
65          ユニットの駆動角度
66
67      Notes
68      -----
69      "VVVAAAVVVAAA;"の13Bytesのメッセージを送信する
70          VVV : 駆動速度
71          AAA : 駆動角度
72          ; : 終端文字
73      """
74      data = bytes(velocity, 'utf-8') + bytes(angle, 'utf-8') + bytes(velocity, 'utf-8') + bytes
            (angle, 'utf-8') + bytes(';', 'utf-8')
75      ser.write(data)
76      # ser2.write(data)
77      print(data)
78      return None
79
80
81  if __name__ == '__main__':
82      # シリアルポートの設定
83      ser = serial.Serial('COM7','9600', timeout=1)
84      # ser2 = serial.Serial('COM6','9600', timeout=1)
85
86      # 1秒待機してユニットを初期化するメッセージを送信
87      time.sleep(1.0)
88      data = bytes('000000000000;', 'utf-8')
89      ser.write(data)
90      # ser2.write(data)
91      print(data)
92
93      # 極座標グラフのセットアップ
94      fig = plt.figure(figsize=(7, 8))
95      ax = fig.add_axes([0.1, 0.2, 0.8, 0.7], projection='polar')
96      ax.set_rlim(0, 255.0)
97      ax.grid(True)
98
99      # クリック位置のプロットを描画
100     point, = ax.plot([0, 0], [0, 0], marker='.', color='red')
101
```

```
    # Reset, Closeボタンのセットアップ
    reset_button_pos = fig.add_axes([0.71, 0.05, 0.09, 0.03])
    reset_button = Button(reset_button_pos, 'Reset')
    close_button_pos = fig.add_axes([0.81, 0.05, 0.09, 0.03])
    close_button = Button(close_button_pos, 'Close')

    # Reset, Closeボタンが押された時の処理
    reset_button.on_clicked(reset)
    close_button.on_clicked(close)

    # グラフ上をクリックした時の処理
    cid = fig.canvas.mpl_connect('button_press_event', onclick)

    plt.show()
```

### A.2.2 Arduino

ソースコード A.4 serial_controller.ino

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

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

#include  <Servo.h>
Servo    svm0,svm1;

char m[48];                                  // 文字列格納用
int i = 0;                                      // 文字数のカウンタ

void setup()
{
    pinMode(3,OUTPUT);      // EnaA
    pinMode(4,OUTPUT);      // INA1
    pinMode(5,OUTPUT);      // INA2
    pinMode(6,OUTPUT);      // EnaB
```

```
42      pinMode(7,OUTPUT);       // INB1
43      pinMode(8,OUTPUT);       // INB2
44
45      pinMode(12,OUTPUT);      // Servo0
46      pinMode(13,OUTPUT);      // Servo1
47
48      svm0.attach(12);
49
50      svm1.attach(13);
51
52      analogWrite(3,0);   // DC0
53      digitalWrite(4,HIGH);
54      digitalWrite(5,0);
55
56      analogWrite(6,0);   // DC1
57      digitalWrite(7,HIGH);
58      digitalWrite(8,0);
59
60     // Serial.begin(9600);
61      Serial.begin(9600);
62  }
63
64  void loop()
65  {
66      int dc0,dc1,sv0,sv1;
67      String input;
68      int len;
69      char msg;
70      int hoge;
71
72      // データ受信したとき
73      if (Serial.available() > 0)
74      {
75          input = Serial.readStringUntil(';');
76          len = input.length(); // debug
77
78          dc0 = (input.charAt(0)-0x30)*100 + (input.charAt(1)-0x30)*10 + (input.charAt(2)-0x30);
79          sv0 = (input.charAt(3)-0x30)*100 + (input.charAt(4)-0x30)*10 + (input.charAt(5)-0x30);
80          dc1 = (input.charAt(6)-0x30)*100 + (input.charAt(7)-0x30)*10 + (input.charAt(8)-0x30);
81          sv1 = (input.charAt(9)-0x30)*100 + (input.charAt(10)-0x30)*10 + (input.charAt(11)-0x30
                );
82
83          analogWrite(3,dc0);
84          analogWrite(6,dc1);
85          svm0.write(sv0);
86          svm1.write(sv1);
87      }
88  }
```

## A.3　第5章で作成したソースコード

CAN 通信を用いた 1 対多数のコントローラの開発で作成したプログラム。

array_id_check.py や can_controller.py、array_id_check.py が保存されているディレクトリに my_module フォルダを作成し、その中に my_skewt.py(ソースコード A.1) を保存する。

#### 動作環境

- Python v3.10.5
- pandas v1.4.3
- numpy v1.23.1
- matplotlib v3.5.2
- pyserial v 3.5

### A.3.1　csv

ソースコード A.5 array_config.csv

```
ID,position_x,position_y
00010001,1,1
00100001,2,1
00110001,3,1
01000001,4,1
00010010,1,2
00100010,2,2
00110010,3,2
```

### A.3.2　Python

ソースコード A.6 can_controller.py

```
#!/usr/bin/env python3
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

from my_module import my_skewt

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
    on_route : bool
        内外判定の結果
    on_chaged : bool
        変更があるかどうかの判定用
    """
    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([1.0/3.0, -1.0/3.0, -2.0/3.0, -1.0/3.0, 1.0/3.0, 2.0/3.0,
            1.0/3.0])
        hex_profile_y = np.array([1.0/3.0, 2.0/3.0, 1.0/3.0, -1.0/3.0, -2.0/3.0, -1.0/3.0,
            1.0/3.0])

        # 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]])

        # 経路上にあるかどうかの判定用
        self.on_route = False

        # 変更があるかどうかの判定用
        self.on_changed = False

    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"""
```

```
        self.on_route = True
        if self.velocity != velocity:
            self.velocity = velocity
            self.on_change = True
        if self.angle != angle:
            self.angle = angle
            self.on_changed = True
        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.on_route = False
        self.velocity = 0
        self.angle = 90
        self.on_changed = True
        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.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 cosθ + y cosφ
```

```
        y' = x sinθ + y sinφ
    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' sinφ - y' cosφ)
        y = 1/det(T)*(-x' sinθ + y' cosθ)
    ただしdet(T)=cosθsinφ-cosφsinθ, 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
    """
    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])
    fig.canvas.draw_idle()
    return None

def motion(event) -> None:
    """グラフ上でマウスを動かした時に実行する関数.

    Notes
    -----
    マウスカーソル位置を現在地に設定し、グラフの描画を更新する
    作動させるユニットを特定してメッセージを送信する
    """
    mouse_position = np.array([event.xdata, event.ydata])
    # マウスカーソルの位置がプロット範囲内にあるときプロットを更新する
    if (mouse_position[0] != None) and (mouse_position[1] != None):
        point.set_data([mouse_position[0], destination_point[0]], [mouse_position[1],
            destination_point[1]])
        for unit_i in array.unit:
            # 内外判定を実行し、
                Trueなら速度と角度をユニットに渡し、変更がある時メッセージを送信する
            if mtplpath.Path(unit_i.contour).contains_point([mouse_position[0], mouse_position
                [1]]) == True:
                mouse_position_cartesian = np.array(oblique2cartesian(mouse_position[0],
                    mouse_position[1]))
                angle = get_angle_vector(mouse_position_cartesian, destination_point_cartesian
                    )
                velocity = 200
                unit_i.set_on_route(round(velocity), round(angle))
                if unit_i.on_changed == True:
                    message = get_message(unit_i.id, unit_i.velocity, unit_i.angle)
                    time.sleep(0.1)
```

```
                    print(message)  # debug
                    ser.write(message)
                    unit_i.on_changed = False
            # 内外判定でFalseの時、オン状態のユニットをオフにし原点回帰する
            else:
                if unit_i.on_route == True:
                    unit_i.set_off()
                    message = get_message(unit_i.id, unit_i.velocity, unit_i.angle)
                    time.sleep(0.1)
                    print(message)  # debug
                    ser.write(message)
                    unit_i.on_chanaged = False
    fig.canvas.draw_idle()
    return None


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("COM3", "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.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
    point, = plt.plot([], [], marker="x", color="red")
    goal, = plt.plot(destination_point[0], destination_point[1], marker="$GOAL$", markersize
        =25)

    # 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)

    # グラフの表示
    plt.show()
```

### A.3.3 Arduino ジャンクション基板

ソースコード A.7 junction.ino

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

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

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

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

void setup(){
    Serial.begin(9600);

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

    if (CAN0.begin(MCP_ANY, CAN_125KBPS, MCP_20MHZ) == CAN_OK)
        digitalWrite(19,0);
    else
        digitalWrite(19,1);

    CAN0.setMode(MCP_NORMAL);
}

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

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

    // データ受信したとき
    if(Serial.available()){
        input = Serial.readStringUntil(';');

        idd = (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;

        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;

        // 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]=(char)dc;        // DC^^ef^^be^^93^^ef^^bd^^b0^^ef^^be^^80の値
        // buf[1]=dc;       // DC^^ef^^be^^93^^ef^^bd^^b0^^ef^^be^^80の値
        buf[2]=(char)sv;        // Servo^^ef^^be^^93^^ef^^bd^^b0^^ef^^be^^80の値
        // buf[2]=sv;       // Servo^^ef^^be^^93^^ef^^bd^^b0^^ef^^be^^80の値
        // buf[3]=0;               // ^^ef^^be^^98^^ef^^bd^^bb^^ef^^be^^9e^^ef^^bd^^b0^^ef^^be^^8c^^ef^^be^^9e  未使用
        if(input.charAt(8) == 0x31){
            buf[3] = (char)1;
        }
        else{
            buf[3] = (char)0;
        }
        buf[4]=0;               // ^^ef^^be^^98^^ef^^bd^^bb^^ef^^be^^9e^^ef^^bd^^b0^^ef^^be^^8c^^ef^^be^^9e  未使用
        buf[5]=0;               // ^^ef^^be^^98^^ef^^bd^^bb^^ef^^be^^9e^^ef^^bd^^b0^^ef^^be^^8c^^ef^^be^^9e  未使用
        buf[6]=0;               // ^^ef^^be^^98^^ef^^bd^^bb^^ef^^be^^9e^^ef^^bd^^b0^^ef^^be^^8c^^ef^^be^^9e  未使用
        buf[7]=0;               // ^^ef^^be^^98^^ef^^bd^^bb^^ef^^be^^9e^^ef^^bd^^b0^^ef^^be^^8c^^ef^^be^^9e  未使用

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

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

    }

    // *** センサ値の受信 ***
    if(!digitalRead(CAN0_INT))                    // If CAN0_INT pin is low, read receive buffer
    {
        CAN0.readMsgBuf(&id, &len, buf);           // Read data: len = data length, buf = data byte(s)
        if((id & 0x80000000) == 0x80000000)                         // 0x1FFFFFFF => 29bit
            sprintf(m, "Extended ID: 0x%08X  DLC: %1d  Data:", (id & 0x1FFFFFFF), len);
        else
            sprintf(m, "Standard ID: 0x%04X  DLC: %1d  Data:", id, len);
```

```
123
124 //         Serial.print(m);
125
126         if((id & 0x40000000) == 0x40000000)       // Determine if message is a remote request
                frame.
127         {
128             sprintf(m, " REMOTE REQUEST FRAME");
129 //            Serial.println(m);
130         }
131         else
132         {
133             rid=(unsigned char)(id & 0x00ff);
134             if (rid==0)                                 // ^^ef^^bd^^bc^^ef^^be^^9e^^ef^^bd^^ac^^
                    ef^^be^^9d^^ef^^bd^^b8^^ef^^bd^^bc^^ef^^bd^^ae^^ef^^be^^9d基板のIDは0
135             {
136                 sprintf(m,"id=%d adc=%d",buf[0],buf[1]);
137                 Serial.println(m);
138             }
139         }
140     }
141 }
```

### A.3.4 Arduino ドライバ基板

ソースコード A.8 driver.ino

```
1  // *****  simp125.ino  10191217  *****
2
3  /*
4   * D0   TX0
5   * D1   RX0
6   * D2   INT
7   * D3   INA1
8   * D4   INA2
9   * D5   servo
10  * D6   DipSW0
11  * D7   DipSW1
12  * D8   DipSW2
13  * D9   DipSW3
14  * D10  CS
15  * D11  MOSI
16  * D12  MISO
17  * D13  SCK
18  * A0 (D14)   ^^ef^^be^^8c^^ef^^bd^^ab^^ef^^be^^84^^ef^^bd^^be^^ef^^be^^9d^^ef^^bd^^
        bbの値を入力
19  * A1 (D15)   ^^ef^^be^^8c^^ef^^bd^^ab^^ef^^be^^84^^ef^^bd^^be^^ef^^be^^9d^^ef^^bd^^
        bbへ電源の供給
20  * A2 (D16)   DipSW4
21  * A3 (D17)   DipSW5
22  * A4 (D18)   DipSW6
23  * A5 (D19)   DipSW7
24  * A6 (D20)
25  * A7 (D21)
26  */
27 #include <stdio.h>
28 #include <mcp_can.h>
29 #include <SPI.h>
30 #define  CAN0_INT 2                              // Set INT to pin  2
31 MCP_CAN CAN0(10);                                // Set  CS to pin 10
32
```

```
#include  <Servo.h>
Servo    svm0;

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

unsigned char      rid,sid;             // rid 受信id   sid ^^ef^^bd^^bd^^ef^^bd^^b2^^ef^^bd^^af^^ef^^be^^81id

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(14,INPUT);      // sens入力
    pinMode(15,OUTPUT);     // ^^ef^^be^^8c^^ef^^bd^^ab^^ef^^be^^84^^ef^^bd^^be^^ef^^be^^9d^^ef^^bd^^bbのLED電源

    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);         // ^^ef^^be^^8e^^ef^^bd^^b0^^ef^^be^^91^^ef^^be^^8e^^ef^^be^^9f^^ef^^bd^^bc^^ef^^be^^9e^^ef^^bd^^bc^^ef^^bd^^ae^^ef^^be^^9d
    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;

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

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

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

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

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

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

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

            if (sid == rid)
            {
                // サーボモータの角度は半分で送られてくるので2倍する。
                dc=buf[1]*2;           // DCモータのPWM値
                sv=buf[2]*2;           // サーボモータの角
                // direction = buf[3];

                // for(;;){

                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);
                }

```

```
158                 svm0.write(sv);
159
160                 // if(Serial.available() > 0)
161                 // {break;}
162                 // }
163             }
164         }
165     }
166     delay(100);
167 }
```