

令和 5 年度 卒業論文

筋電義手人工感覚のための駆動部エネルギー  
測定に基づく力覚センシング

Kinesthetic sense based on evaluation of actuator energy for  
artificial sense in myoelectric prosthetic hands

北海道大学

工学部情報エレクトロニクス学科電気電子工学コース  
量子知能デバイス研究室 学部 4 年

半田 寛明

指導教員 葛西 誠也 教授

# 目次

第 1 章 序論 .....	1
1.1 背景 .....	1
1.2 目的 .....	2
1.3 本論文の構成 .....	3
第 2 章 力覚センシング .....	4
2.1 既存の力覚センシング技術 .....	4
2.1.1 歪みゲージセンサ .....	4
2.1.2 圧電センサ .....	4
2.1.3 静電容量センサ .....	4
2.1.4 光学センサ .....	4
2.1.5 サーボの逆起電力とサーボ内蔵センサを用いた手法 .....	5
2.1.5.1 センサレスセンシングの数理的説明 .....	5
2.2 既存技術の課題 .....	5
2.2.1 力覚センサの問題点 .....	6
2.2.2 サーボの逆起電力を用いた場合の問題点 .....	6
2.3 サーボの消費電力を用いる方法 .....	6
第 3 章 筋電義手の駆動部エネルギー測定について .....	8
3.1.1 実験器具 .....	8
3.2 実験方法 .....	8
3.2.1 手順 .....	10
3.3 実験結果 .....	10
第 4 章 リザバーコンピューティング .....	23
4.1 概要 .....	23
4.2 評価方法 .....	24
4.2.1 RMSE (平方平均二乗誤差) .....	24

第 5 章 消費電力から負荷を学習 .....	25
5.1 データの前処理 .....	25
5.2 リザバーコンピューティングの各種設定 .....	27
5.2.1 リザバー層 .....	27
5.2.2 リードアウト層 .....	27
5.3 学習結果 .....	27
5.4 考察 .....	30
第 6 章 まとめ .....	31
謝辞 .....	32
Appendix .....	33
参考文献 .....	50

# 第 1 章 序論

## 1.1 背景

何らかの事故や病気により後天的に上肢を無くしてしまった人、そして先天的に上肢がない人は本人の意思とは無関係に、取れる社会参加の選択肢が限られてしまう。社会構造自体をそのような人々も参画しやすい形に変革する方法も考えられるが、そのためには上肢を必要としないコミュニケーションや業務執行の方法を考える必要が出てくるため、いずれにせよ技術的課題に直面する。

このため、上肢を補う形での社会参加が克服すべきハードルが最小に思える。そこで使用者自身の身体の一部のように感じることのできる操作感の義手が必要になってくるのは明らかである。そもそも義手には審美性のみを目的とした物も存在するが、操作可能な上、侵襲性がないため着脱が容易な筋電義手が本分野のこれから潮流を為すと考えられる。

人間が自分の手を動作させる際、随意動作（意識的動作）と不随意動作（無意識的動作）に分類できる。随意動作とは、人間が意図した通りの動きをする動作である一方、不随意動作は人間が動作に対して意識的にならずとも確かに実行している動作や制御のことである。熱いものや痛みを感じると同時にむしろ感じるよりも前に手を引っ込めたり、重いものを持つときに無意識に力を加えてものが落ちないように制御したりといったことがこれに当たる。筋電義手の目指す先は使用者があたかも自分の手であったかのように操作を行える状態である。このため、その完成度は問題になるが、筋電義手における随意操作性能自体は研究分野としてすでに大きなものとして存在する[1]。一方で不随意の動作に関してはその課題感の共有の難しさから課題として捉えられにくい分野になってくる。しかし、この不随意動作無くして人の手のような操作感は得られない。

Technology	Most widespread	Mature, home use	Cutting edge, laboratory use	Future directions
Interface	Body harness <sup>c</sup>	sEMG <sup>a</sup> iEMG (51) <sup>a</sup> Vibrotactile interface (178) <sup>b</sup> TENS <sup>b</sup> FINEs (136) <sup>b</sup> Osseointegration (138) <sup>c</sup>	HD-sEMG (179) <sup>a</sup> Regenerative electrodes (39) <sup>a</sup> TIMEs (6, 42) <sup>b</sup> LIFEs (43) <sup>b</sup> Sieve electrodes (180) <sup>b</sup> Utah Array (113) <sup>c</sup>	Noninvasive intraneuronal stimulation (ultrasound) <sup>b</sup> Soft neurotechnology (48) <sup>b</sup>
Motor decoding	Body power Threshold-based sEMG	EMG-based pattern recognition (e.g., Ottobock Myo Plus, Coapt Gen2)	Simultaneous single-finger classification (75) Linear regression and shared control (79)	Advanced control using regenerative peripheral nerve interfaces (39) Deep learning for single-finger proportional control
Sensory feedback	No feedback	Vibrotactile haptic feedback (129) Touch contact (138) Position (136)	Neuromorphic (62) Texture (61) Object stiffness (6) Multimodal (position and tactile) (64) Biomimetic stimulation (62)	Temperature feedback Proprioception
Sensorization	No skin	Force sensors (measuring motor current) Sensorized fingertips (e.g., bebionic)	Asynchronously coded electronic skin (125)	Soft embedded sensors (115) Bioinspired flexible organic artificial afferent nerve (122)

Abbreviations: EMG, electromyography; FINE, flat interface nerve electrode; HD-sEMG, high-density surface electromyography; iEMG, implanted EMG; LIFE, longitudinal intrafascicular electrode; sEMG, surface electromyography; TENS, transcutaneous electrical nerve stimulation; TIME, transverse intrafascicular multichannel electrode.

<sup>a</sup>Motor interface.

<sup>b</sup>Sensory interface.

<sup>c</sup>Both motor and sensory interface.

表 1: 筋電義手研究の方針と現在地

## 1.2 目的

不随意動作を暗黙的に義手に実装するのでは自分の手のようにはならない。生体の手のように、触る温度の形状、触り心地などはもちろんのこと、手にかかっている重量的な負担をも利用者にフィードバックする必要がある。触覚に当たる部分を皮膚感覚、重量的負担を感じる力覚の部分を深部感覚と呼ばれているが、皮膚感覚の分野については義手のみならず xR などの分野でもハプティクスと呼ばれるように、他分野に渡り研究が進められている。一方で深部感覚については皮膚感覚と同様に重要であるにも関わらず比較的研究が進められていない。これは Google Scholar にて “haptics” は 225,000 件、“cutaneous sensation” は 516,000 件ヒットするのに対し、深部感覚にあたる “bathymesthesia” は 86 件しかヒットしないことからも明らかである。なお “deep sense” や “deep sensing” などはディープラーニングのアルゴリズムなど多岐にわたる分野での使用がされている用語であるため除外した。これらの研究動向を踏まえ、我々は重要にも関わ

らずあまり手をつけられていない分野である力覚のフィードバックに今回取り組むこととした。

### 1.3 本論文の構成

本論文は全 6 章から構成されている。

第 1 章では本論文の目的及び先行研究を踏まえた立ち位置を論じた。

第 2 章では力覚センシング技術を紹介する。

第 3 章では筋電義手の駆動部エネルギー測定することで如何に力覚センシングを行うかを説明する。

第 4 章では本研究結果を用いて推論モデルを作成するのに用いるリザバーコンピューティングの説明を行う。

第 5 章では駆動部エネルギーという信号をリザバーコンピューティングを用いて解析する手法について紹介する。

最後に第 6 章を本論文のまとめとする。

# 第2章 力覚センシング

力覚を測定するための機構は古くから研究されており、産業用ロボットやタッチディスプレイなど幅広い用途で利用してきた。以下に主な力覚センシングをする技術を紹介する。 [2]

## 2.1 既存の力覚センシング技術

### 2.1.1 歪みゲージセンサ

歪みゲージセンサはセンサー部にかかる引張力・圧縮力により、電気抵抗が変化する金属抵抗材料の性質を利用して、力やトルクに換算する。小型で精度が高く、応答性も高いことから多くの場面で使われる。

### 2.1.2 圧電センサ

水晶やPZT(ジルコン酸チタン酸鉛)などの圧電効果を有する材料をセンサー部に使用して、力を測定する。小型で応答性が高く、コストも比較的優れいるが精度は、ひずみゲージ式や静電容量式には及ばない。

### 2.1.3 静電容量センサ

センサー部を金属材料の電極が向かい合わせに配置されたコンデンサ型とし、力により導体間にひずみが生じて距離が変わることによる静電容量の変化を検知する。構成が比較的簡単で低成本で生産できる上、電極をフィルム状にすると、小型化・薄型化が可能である。精度や応答性も優れている。

### 2.1.4 光学センサ

計測対象物に一定間隔で模様をマーキングしておき、力が加わった時に生じる模様の変化を、カメラやレーザーなどの光学センサで検出して、力の大きさを計算して求める。非接触で測定できることが最大のメリットではあるが精度、応答性、小型化、コストは、他の方式に軍配が上がるため使われる場面は限られる。

### 2.1.5 サーボの逆起電力とサーボ内蔵センサを用いた手法

先述のようなセンサを用いない手法も古くから研究されている。特に今回のようにサーボモータを内蔵する場合が非常に多い筋電義手のようなものの場合、もともともと内蔵されているサーボから得られる信号を用いる方法として逆起電力を観測する方法がある。これはモータに外力を与えて回転させた際にモータが発電機として振る舞う現象を用いる手法である。しかしこれだけでは回転方向がわからぬため、サーボモータの回転角制御用に搭載されているエンコーダやタコジエネレータを用いることで外力の方向を割り出すというものである。

以下にその数理的説明を行う。

#### 2.1.5.1 センサレスセンシングの数理的説明

全外乱から外力以外の要素をモデル化し取り除くことで外力のみを抽出可能として考えると以下の式で表現される。

$$F_{\text{ext}} = F_{\text{dis}} - (F_{\text{int}} + F_g + F_c + DsX^{\text{res}} + \Delta Ms^2 X^{\text{res}} - \Delta K_t I_a^{\text{ref}}) \quad (1)$$

ただしここで  $F_{\text{dis}}$  は全外乱、 $F_{\text{int}}$  はコリオリ力・遠心力などの内部干渉力、 $F_g$  は姿勢の影響で重力加速度により発生する力、 $F_c$  はクーロン摩擦力、 $DsX^{\text{res}}$  は粘性摩擦力、 $M$  はモータ稼働部の慣性であり  $Ms^2 X^{\text{res}}$  は周波数領域におけるモータの動力学を表す(すなわち  $s, X^{\text{res}}$  はそれぞれラプラス演算子と周波数領域での角度を表す変数)。 $K_t$  はモータに使用されている磁石の強さやコイルの巻線量によって変化するモータ固有の推力定数、 $I_a^{\text{ref}}$  は周波数領域における電流を表す変数である。

式(1)における C 摩擦力  $F_c$ 、粘性摩擦力  $DsX^{\text{res}}$  は二種の等速度試験、慣性変動  $\Delta M$  は加速度試験によって同定することができる。[3]

## 2.2 既存技術の課題

力覚を取得する方法には各種センサを用いる方法と、内蔵されているサーボモータから得られる信号を用いる方法とが一般的であることがわかった。しかし、どちらの方法も筋電義手に力覚を与えるためのアプローチとしては相応しくない。

### 2.2.1 力覚センサの問題点

前節で紹介した力覚センサを筋電義手に用いるとなると二つの懸念点が浮かんでくる。搭載センサの取り付け位置とコストの増加だ。前節で紹介した力覚センサはどれも圧力が直接かかる部分に設置する必要がある。物に触れた際に感じる感触が認識対象であればこれでも構わないが、今回対象とするのは稼働部にかかる力覚であるため力覚センサも稼働部に取り付ける必要が出てくる。駆動部にある程度サイズのある力覚センサを取り付けるとなると取り付け位置や取り付け方にかなりの工夫が必要となってくることは想像に難くない。

また、コストの面でも各サーボ関節部分に力覚センサが必要となってくることから効果になってくる。筋電義手は上肢がない方が生活に支障のないようするという、人権の面からも最低限必要なものとなってくるものであるため、低コストに抑えることが重要である。そのためには各関節ごとに高価なセンサを搭載することは望ましくない。

### 2.2.2 サーボの逆起電力を用いた場合の問題点

既存技術の中で安価なアプローチとしてサーボの逆起電力とエンコーダ等の回転方向を検出するセンサとを組み合わせたセンサレスの手法がある。前節でも紹介した通り、この手法を用いることで、逆起電力を検出する回路以外の追加部品を必要とせず実装できる。また、追加部品が必要ないため取り付け位置に創意工夫の必要性がない。

しかし決定的な弱点が存在する。モータの特性上、逆起電力を得るにはシャフトの回転が必要な点である。実際の力覚を測定する際、モータを回転させるには弱すぎる力が付与される場合も多く存在する。しかし、そのような場合でもこのセンサレスの手法ではかかっている力覚は0として判断されてしまう。これでは実際の生体のように、外力に耐えている状態でも筋肉の負荷を感じ取ることで外力の大きさを大凡認知することは叶わない。

## 2.3 サーボの消費電力を用いる方法

前節では力覚センサの搭載や、サーボの逆起電力を用いた、場合の欠点を紹介した。筋電義手の力覚を測定するためにはこれらの方法では実用性に欠ける部分があることがわかった。これらの方法より簡便かつ静止時の計測も可能な手法が望ましいわけであるが、ここで我々は他の研究テーマでのトラブルに着目した。

### 過負荷によるサーボの故障

同じ研究室内の別の研究グループでサーボの故障が何度か発生した。そのグループではロボットを使用していたのだが、ロボットの足を為すサーボに過負荷がかかり電流制御用のトランジスタが焼き切れてしまうことが原因だった。サーボは与えられた PWM 信号に対応する回転角度（目標角度）と現在の回転角度とにずれがある場合に目標角度となるように回転する性質がある。目標角度まで回転する際に外力により回転にブレーキがかかる際はそれに負けじと供給電流を増やすことで動こうとする。しかしその際に内部の電子回路の許容電流を超えてしまい破損することがあった。我々はこの現象に着目をして、サーボにかかる外力を消費電力を用いて検出する手法を考案した。

# 第3章 筋電義手の駆動部エネルギー測定について

筋電義手の稼働部を為すサーボの消費電力を測定することで、義手にかかる力覚を測定できるというコンセプトの実証を行った。今回は簡単のために筋電義手を用いるのではなく、実験用のロボットアームを用いた。

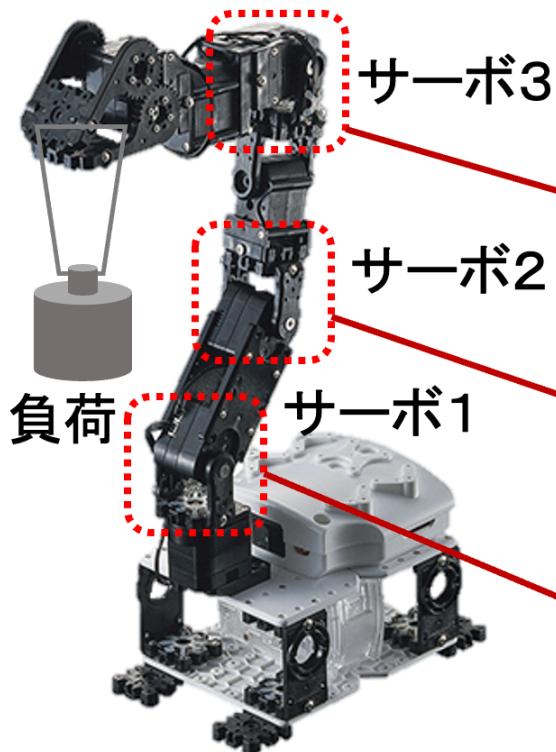
## 3.1.1 実験器具

以下に今回の実験で使用した実験器具を表記する

- ロボットアーム：近藤科学株式会社 KXR-A5
- マイコン：Arduino Uno
- シヤント抵抗： $1\Omega$
- 差動増幅回路
  - オペアンプ：Texas Instrument TLC-2274
  - 抵抗 1:  $1k\Omega$
  - 抵抗 2:  $2k\Omega$
- 分銅： 50g, 100g

## 3.2 実験方法

## ロボットアーム



近藤科学 KXR-A5

## サーボ電力評価回路

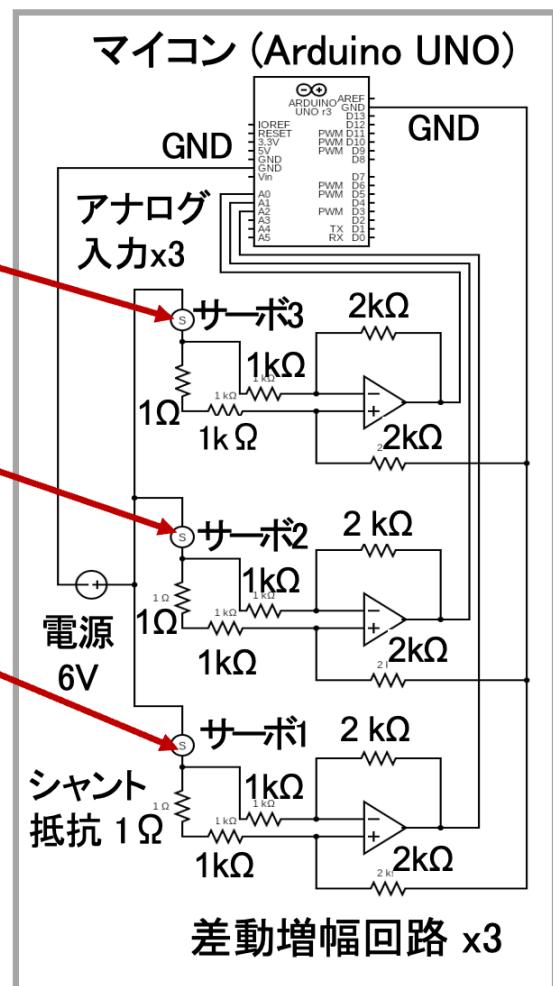


図 1: 実験器具構成図

上図のように筋電義手を模したロボットアームのサーボの3つを使い、それぞれの消費電力を電流を計測することで測定した。各サーボの消費電力は、サーボの電源ラインに直列にシャント抵抗を接続し、その抵抗の始端と終端の電圧を差動增幅回路を用いることでマイコンにアナログ情報として取り入れられるようにした。

計測に用いたマイコンは、各サーボの動作を指令するマイコンと同一のものを用いることでサーボの動作開始とほとんど同時に計測が開始されるように設計した。ここでサーボの動作開始と計測開始とが完全に同時でないのは、今回使用したマイコンでは同時に一つのスレッドしか実行できないため厳密には数クロック分だけ動作開始と計測開始とがずれてしまうためである。

### 3.2.1 手順

本研究の実験では二つの動作パターンのデータをロボットアームに付加する負荷を変えながら取得することで動作パターンと重さの関係性を調べやすくした。以下にその手順を示す。

1. 負荷をかけない状態で動作1を実行し、その際のシャント抵抗にかかる電圧を358Hzで1秒間計測した。これを8回繰り返した。
2. 1と同様のことを、ロボットアームの先端部分に50gの分銅を吊るした状態で実行した。
3. 1と同様のことを、ロボットアームの先端部分に100gの分銅を吊るした状態で実行した。
4. 動作2で1~3と同様のことを実行した。

## 3.3 実験結果

この実験を行った結果、次の図に示すような消費電力の時系列変化が見られた。

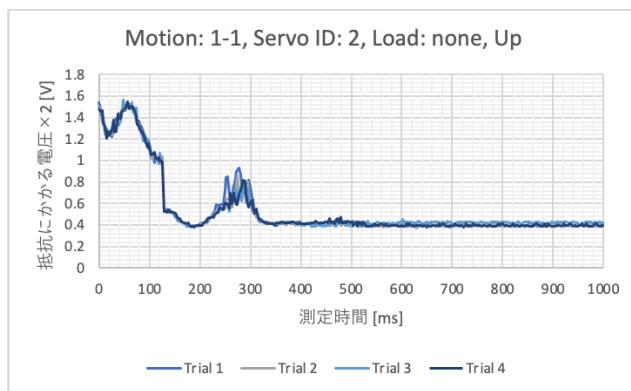


図 2: 動作 1（上昇）の際のサーボ 1 の消費電力（負荷 0g）

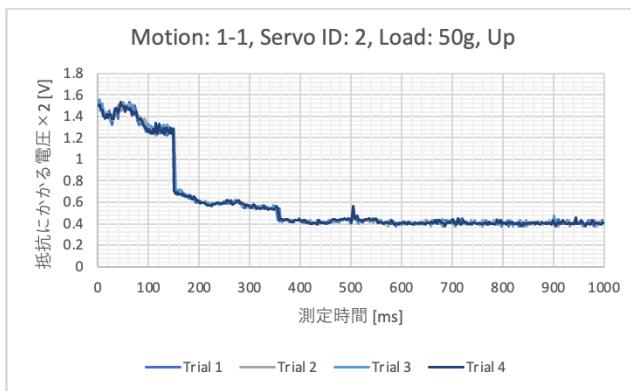


図 3: 動作 1（上昇）の際のサーボ 1 の消費電力（負荷 50g）

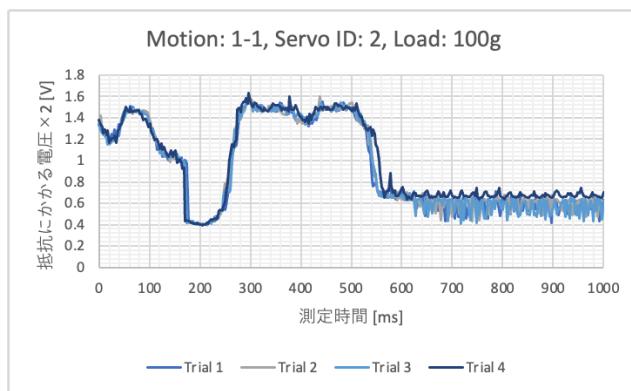


図 4: 動作 1（上昇）の際のサーボ 1 の消費電力（負荷 100g）

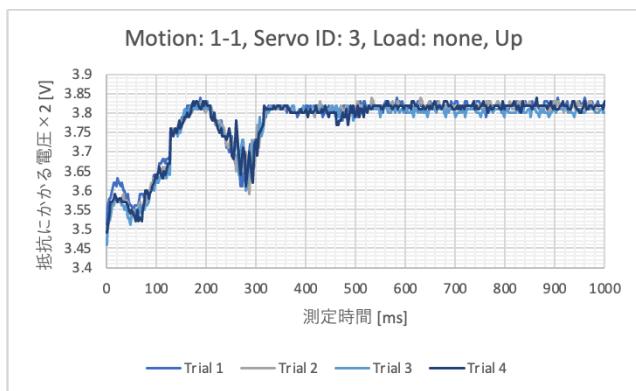


図 5: 動作 1（上昇）の際のサーボ 2 の消費電力（負荷 0g）

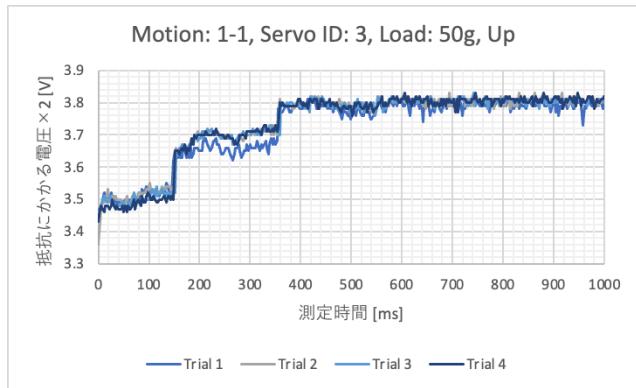


図 6: 動作 1（上昇）の際のサーボ 2 の消費電力（負荷 50g）

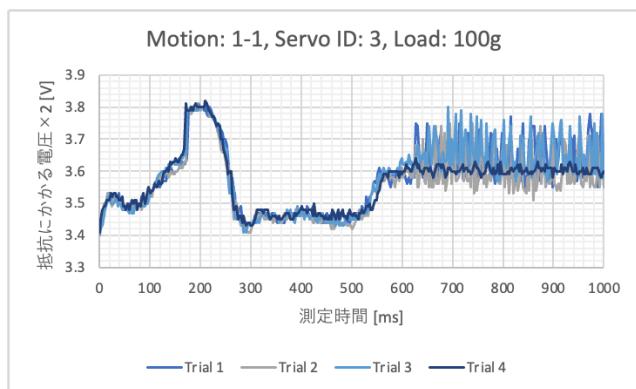


図 7: 動作 1（上昇）の際のサーボ 2 の消費電力（負荷 100g）

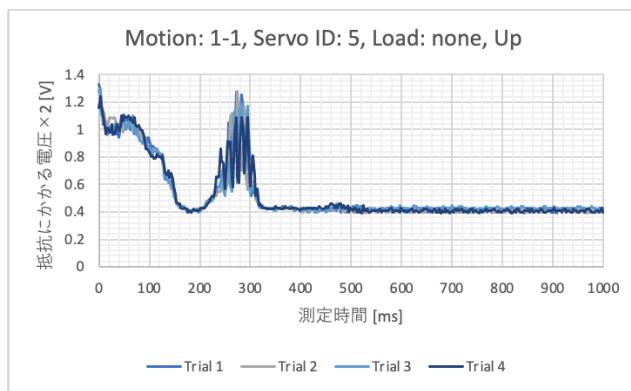


図 8: 動作 1（上昇）の際のサーボ 3 の消費電力（負荷 0g）

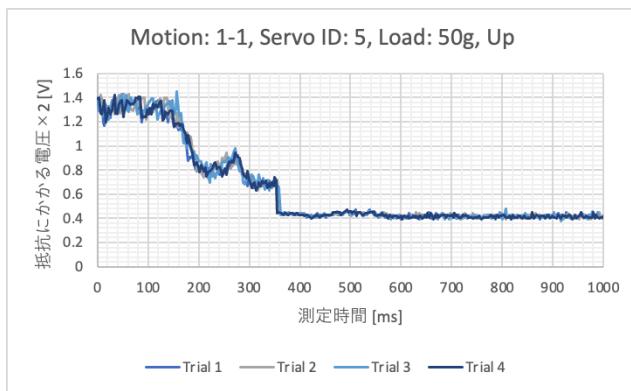


図 9: 動作 1（上昇）の際のサーボ 3 の消費電力（負荷 50g）



図 10: 動作 1（上昇）の際のサーボ 3 の消費電力（負荷 100g）

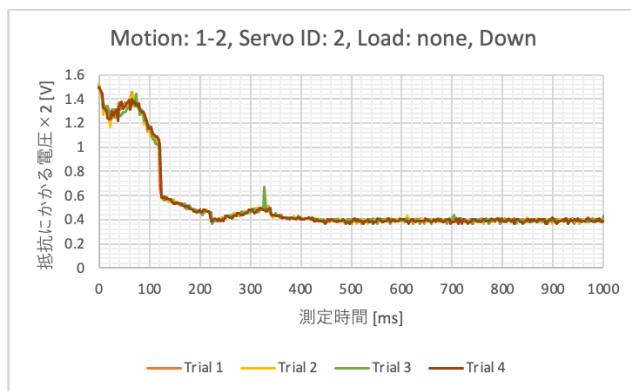


図 11: 動作 1（下降）の際のサーボ 1 の消費電力（負荷 0g）

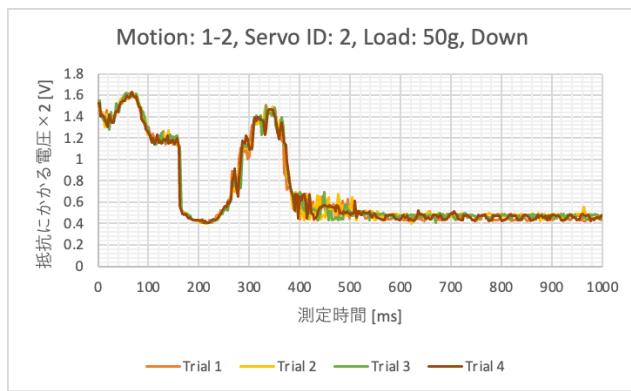


図 12: 動作 1（下降）の際のサーボ 1 の消費電力（負荷 50g）

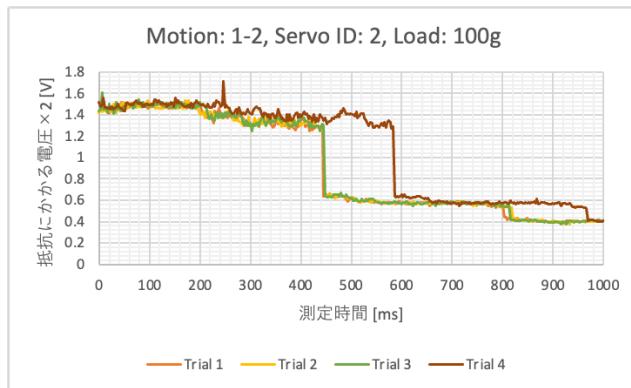


図 13: 動作 1（下降）の際のサーボ 1 の消費電力（負荷 100g）

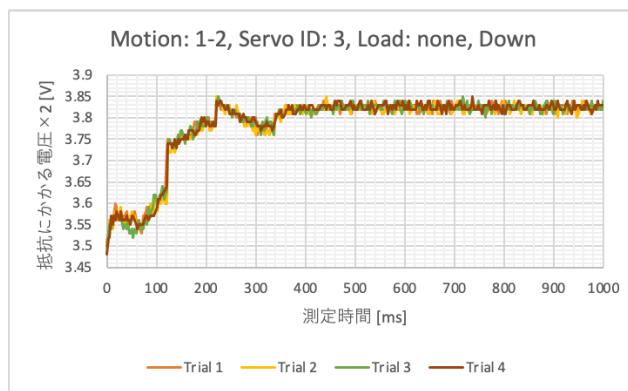


図 14: 動作 1 (下降) の際のサーボ 2 の消費電力 (負荷 0g)

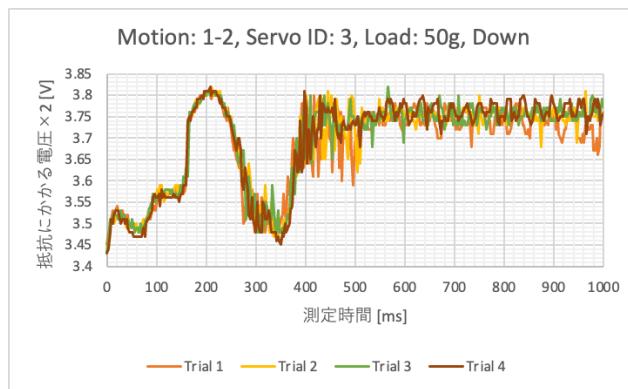


図 15: 動作 1 (下降) の際のサーボ 2 の消費電力 (負荷 50g)

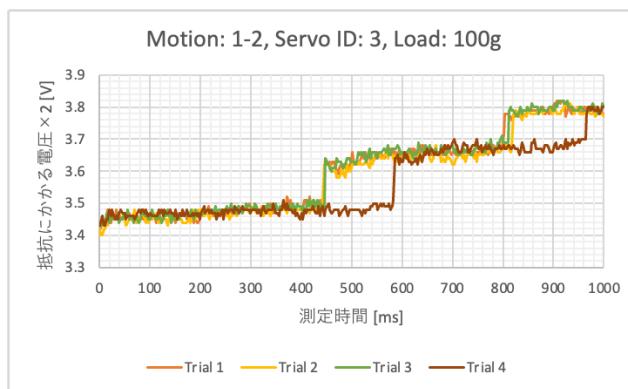


図 16: 動作 1 (下降) の際のサーボ 2 の消費電力 (負荷 100g)

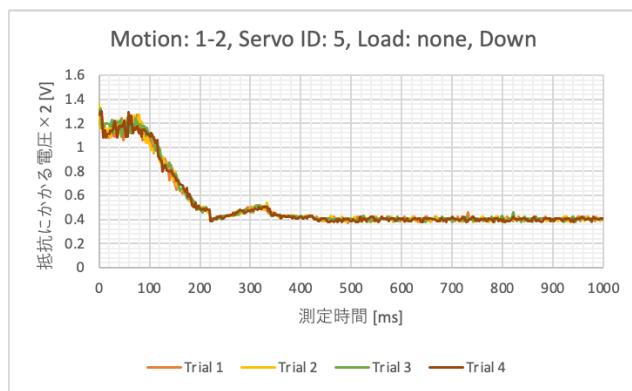


図 17: 動作 1 (下降) の際のサーボ 3 の消費電力 (負荷 0g)

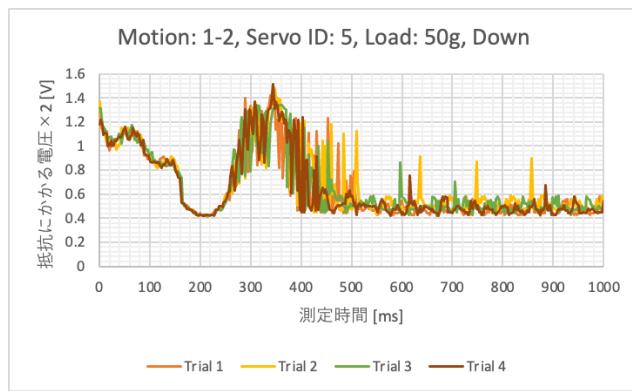


図 18: 動作 1 (下降) の際のサーボ 3 の消費電力 (負荷 50g)

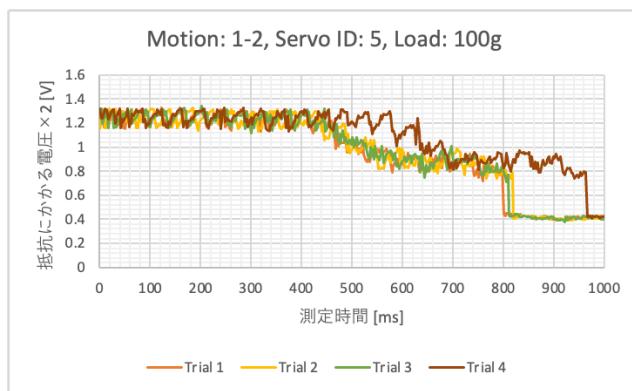


図 19: 動作 1 (下降) の際のサーボ 3 の消費電力 (負荷 100g)

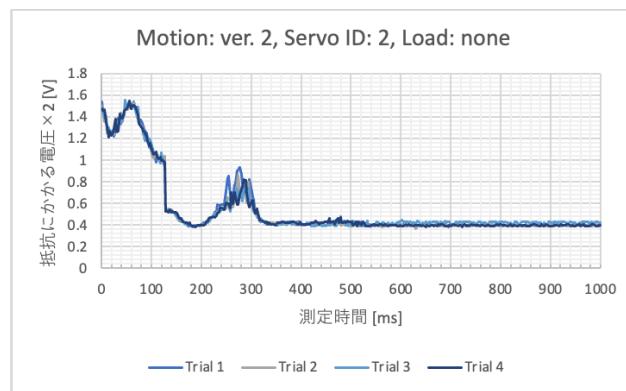


図 20: 動作 2 (上昇) の際のサーボ 1 の消費電力 (負荷 0g)

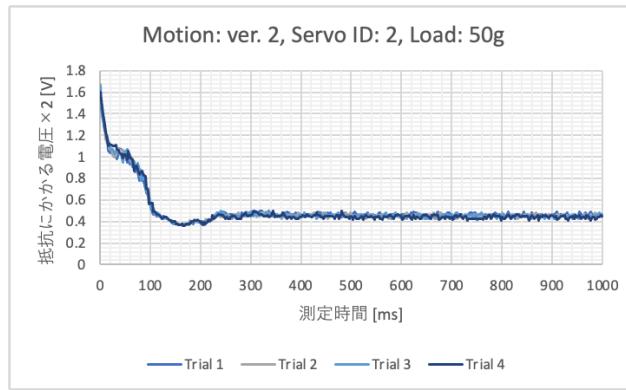


図 21: 動作 2 (上昇) の際のサーボ 1 の消費電力 (負荷 50g)

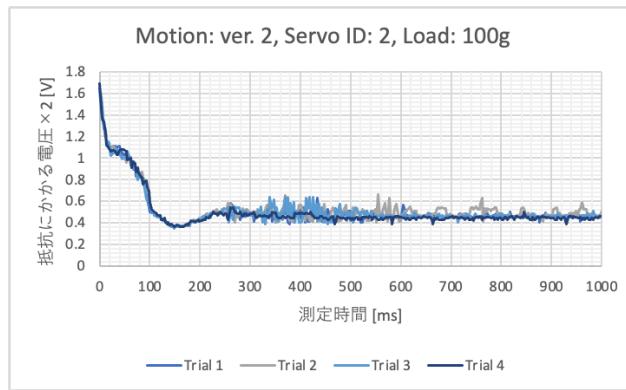


図 22: 動作 2 (上昇) の際のサーボ 1 の消費電力 (負荷 100g)

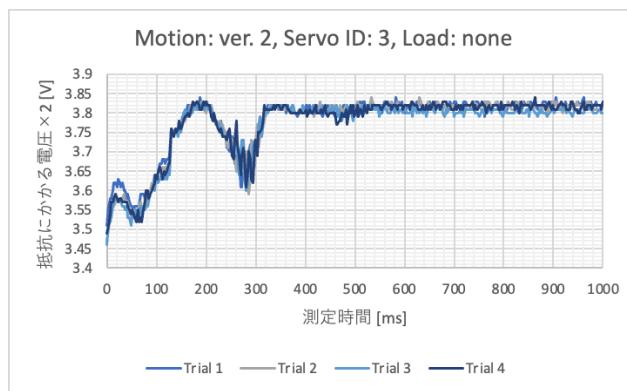


図 23: 動作 2 (上昇) の際のサーボ 2 の消費電力 (負荷 0g)

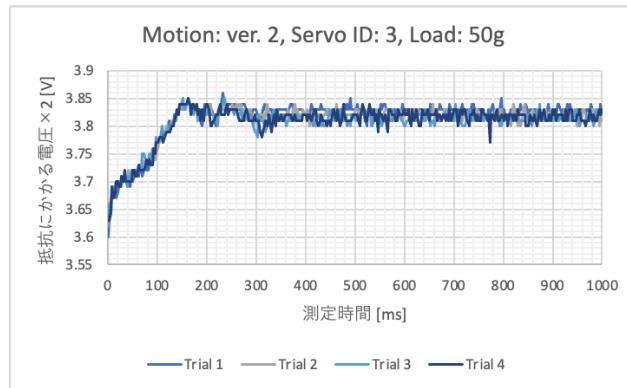


図 24: 動作 2 (上昇) の際のサーボ 2 の消費電力 (負荷 50g)

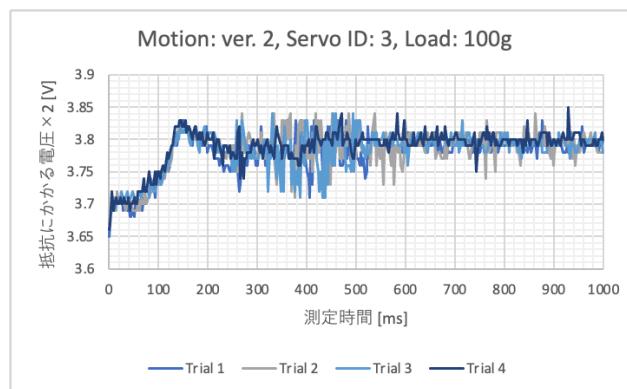


図 25: 動作 2 (上昇) の際のサーボ 2 の消費電力 (負荷 100g)

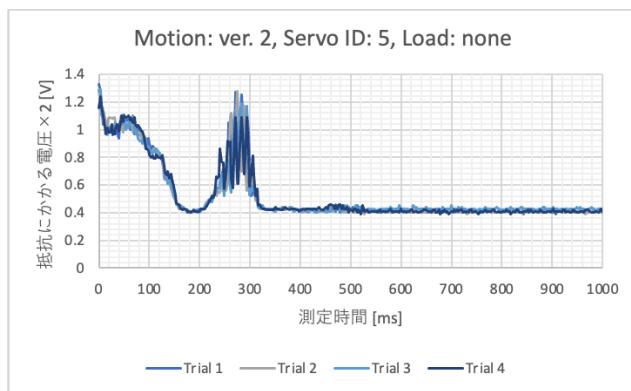


図 26: 動作 2 (上昇) の際のサーボ 3 の消費電力 (負荷 0g)

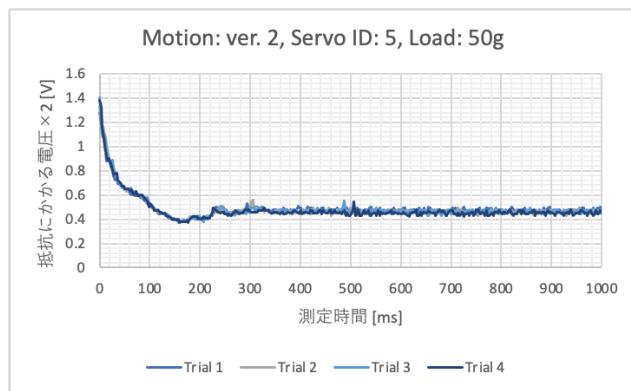


図 27: 動作 2 (上昇) の際のサーボ 3 の消費電力 (負荷 50g)

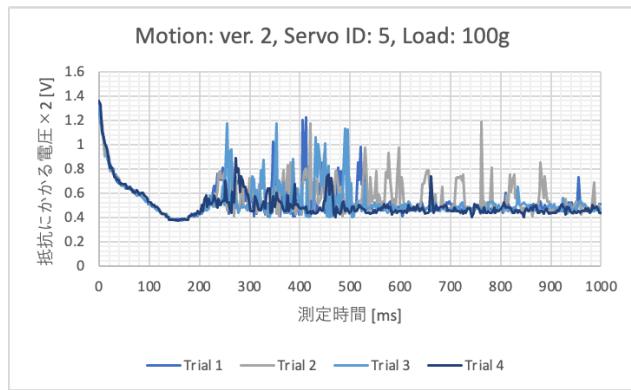


図 28: 動作 2 (上昇) の際のサーボ 3 の消費電力 (負荷 100g)

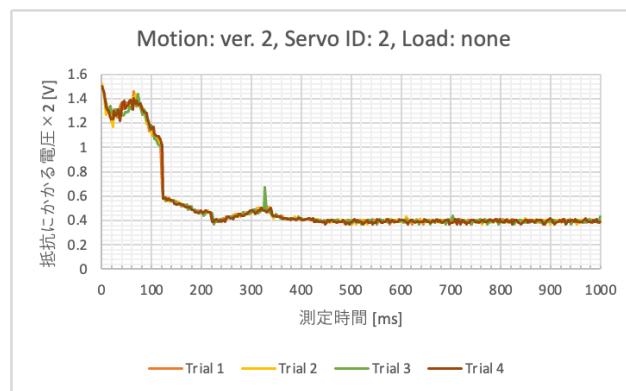


図 29: 動作 2（下降）の際のサーボ 1 の消費電力（負荷 0g）



図 30: 動作 2（下降）の際のサーボ 1 の消費電力（負荷 50g）

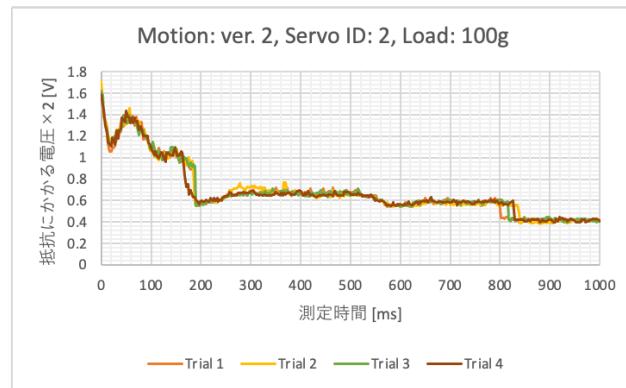


図 31: 動作 2（下降）の際のサーボ 1 の消費電力（負荷 100g）

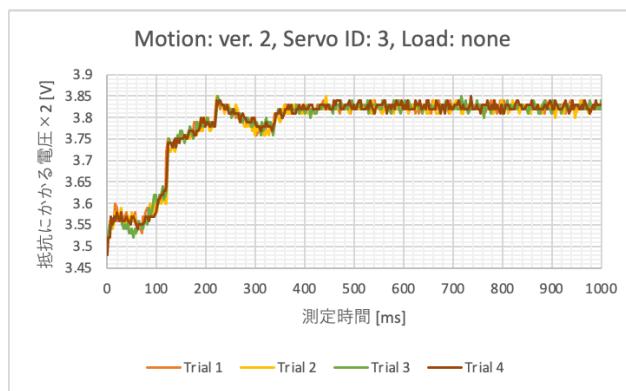


図 32: 動作 2（下降）の際のサーボ 2 の消費電力（負荷 0g）

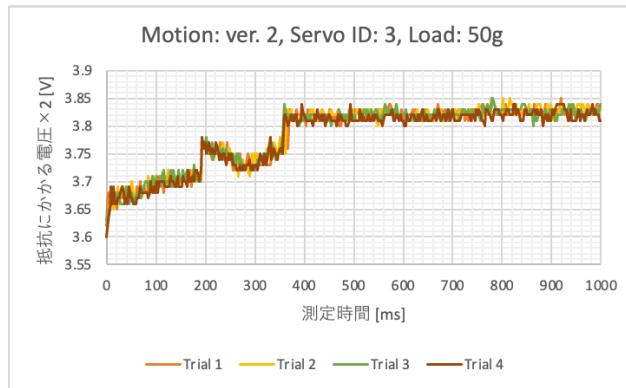


図 33: 動作 2（下降）の際のサーボ 2 の消費電力（負荷 50g）

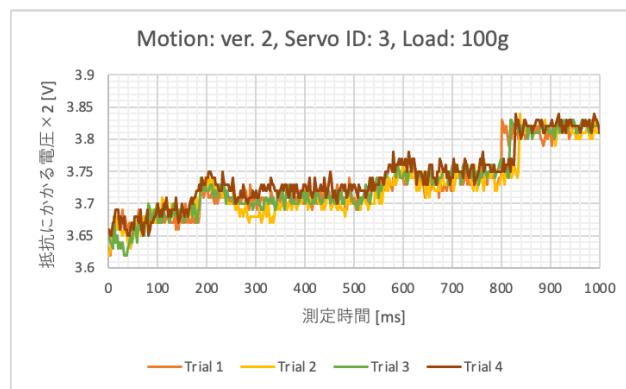


図 34: 動作 2（下降）の際のサーボ 2 の消費電力（負荷 100g）

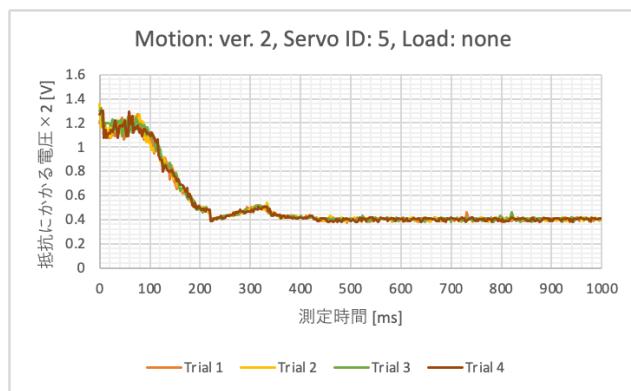


図 35: 動作 2（下降）の際のサーボ 3 の消費電力（負荷 0g）

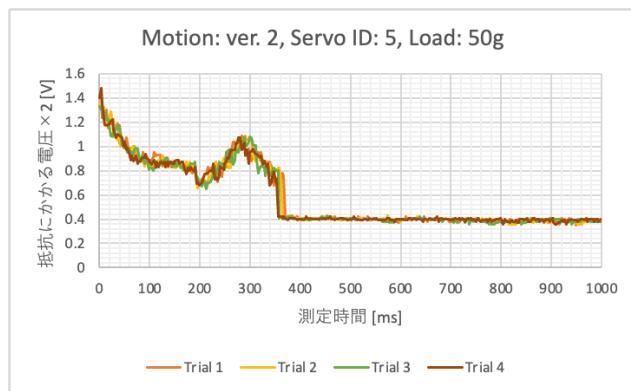


図 36: 動作 2（下降）の際のサーボ 3 の消費電力（負荷 50g）

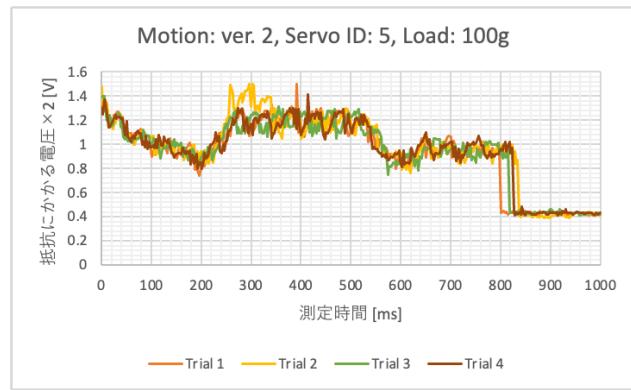


図 37: 動作 2（下降）の際のサーボ 3 の消費電力（負荷 100g）

# 第4章 リザバーコンピューティング

実験で得られた一定の規則性の認められる結果をモデル化することで、入力信号から出力の予測ができるようになる。そのための手法は古くから多く考えられてきたが、計算機の高性能化に伴い近年では機械学習を用いた手法が広く研究されるようになった。機械学習と大きく括ってもそのフレームワークにはいくつもの種類があり、それぞれ得意とする分野が存在する。例えば Convolutional Neural Network を用いた機械学習は画像解析に適しているが時系列データを扱う場面で採用されることは少ない。同様に Transformer は自然言語処理の分野で今や主流となっているが、画像処理には適さない。

## 4.1 概要

前章での結果でも示したとおり、本研究の学習対象は時系列変化する信号になっている。すると時系列変化する信号の学習に適した手法が必要となってくるわけであるが、これにあたるのが Recursive Neural Network (RNN) を用いた機械学習である。しかし従来型の RNN は推論フェーズと実行フェーズともに多くの計算量を必要とし、さらに学習のために多くのデータ数を必要とすることが知られている。学習はともかく、筋電義手を操作するための計算機の処理能力で推論がリアルタイムで行われることが理想的であるため、従来型の RNN のような多くの計算リソースを必要とするようなモデルを採用することは困難となってしまう。[4]

そこで登場するのがリザバーコンピューティング (RC) である。RC は RNN の一種ではあるが、ニューラルネット層の重みづけは行わずに出力層 (リードアウト) の重みづけのみを学習させることでモデルの最適化を行う。この手法を用いることで、多くのデータ数がなくとも十分な精度が得られる学習ができることが先行の研究で明らかになっている。我々は今回 RC を用いてサーボの消費電力

の波形を学習させ、系にかかっている負荷を予測するモデルを作成した。  
RC を用いて得られる結果は

## 4.2 評価方法

モデルの評価には Root Mean Squared Error (RMSE) という指標を用いた。

### 4.2.1 RMSE (平方平均二乗誤差)

モデル出力と目標出力が共に連續値で与えられる時系列予測タスクや、時系列生成タスクの誤差を評価する指標として RMSE が広く用いられる。

$$\text{RMSE} = \sqrt{\frac{1}{T} \sum_{n=1}^T \|\mathbf{d}(n) - \hat{\mathbf{y}}(n)\|_2^2}$$

と定義される。これはモデル出力と目標出力の間の二乗誤差の時間平均の平方根を表している。目標出力データの特徴を考慮していないので、異なるデータに対する RMSE の値を比較に意味がない点は注意されるべきである。また、RMSE はデータや予測値に平均から大きく外れた異常値や外れ値が含まれていると、その影響を大きく受けやすい。[4]

# 第 5 章 消費電力から負荷を学習

3章で取得したデータを使ってRCを用いて学習させることで、3つのサーボの消費電力から系にかかる負荷を割り出すモデルを作成した。また、消費電力のみを与えたところで負荷を予測するモデルを作成したところで汎用性に欠けるという予測のもと、3つのサーボの消費電力の時系列データのほかに特徴量としてそれぞれのサーボの動作開始と動作終了角度、そして設定した回転速度を与えることとした。これらの情報は取得に個別のセンサなどの機構を必要とせずサーボ制御用、すなわちRC推論にも用いるマイコン上で取得可能なデータの中から選んでいる点を強調する。

## 5.1 データの前処理

RCを用いた学習をするためにはまずデータを学習に使用しやすい形にする必要がある。今回はサーボ3つの消費電力の時系列変化という特徴量に加えて、サーボそれぞれの初期角度、終了角度、そして設定速度を横並びにすることデータをわかりやすくした。また、最後にラベルとして機能する負荷重量を記載することで、プログラム上で教師ラベルとして扱いやすい形にした。プログラム上ではこれを48個のファイルに分け、全データ数を48個とし、そのうち38個を訓練データとして、10個を評価用データとしてして使用している。

サンプル番号	サーボ1電力	サーボ2電力	サーボ3電力	サーボ1開始角度	サーボ1終了角度	サーボ1設定速度	サーボ1開始角度	...	サーボ3開始角度	サーボ3終了角度	サーボ3設定速度	ラベル(負荷重量)
1												50g
2												50g
...												...
358												50g

表 2: リザバー学習にをさせる際に用いたデータ構造

## 5.2 リザバーコンピューティングの各種設定

実は RC にもさまざまな方法が提案されている。Echo State Network (ESN), Liquid State Machine (LSM), FORCE などが一般的であるが、今回は最もシンプルで実装の簡単な ESN を採用することとした。ESN は Echo State Property を満たすリザバー層に入力を与え、リードアウトの重みづけを線形回帰を用いて学習させる点から非常にシンプルであるため RC を使用する上で広く使われている。[5]

### 5.2.1 リザバー層

リザバー層には Leaky Integrator によって構成されたニューラルネットを用いた。リーク率は 1、リカレント結合重み行列のスペクトル半径は 0.95、活性化関数にはハイパボリックタンジェントを用いた。ノード数は特徴量の数よりも十分に大きい 100 に設定した。[6]

### 5.2.2 リードアウト層

次にリードアウト層の重みづけの学習方法について説明する。今回は ESN を用いるため線形回帰での重みづけ学習を行ったのだが、外れ値の影響を必要以上に大きくすることを避けるために Ridge 回帰を用いた。正則化パラメータは  $1.0 \times 10^{-3}$  に設定した。[7]

## 5.3 学習結果

以下がこの学習を行った結果である。

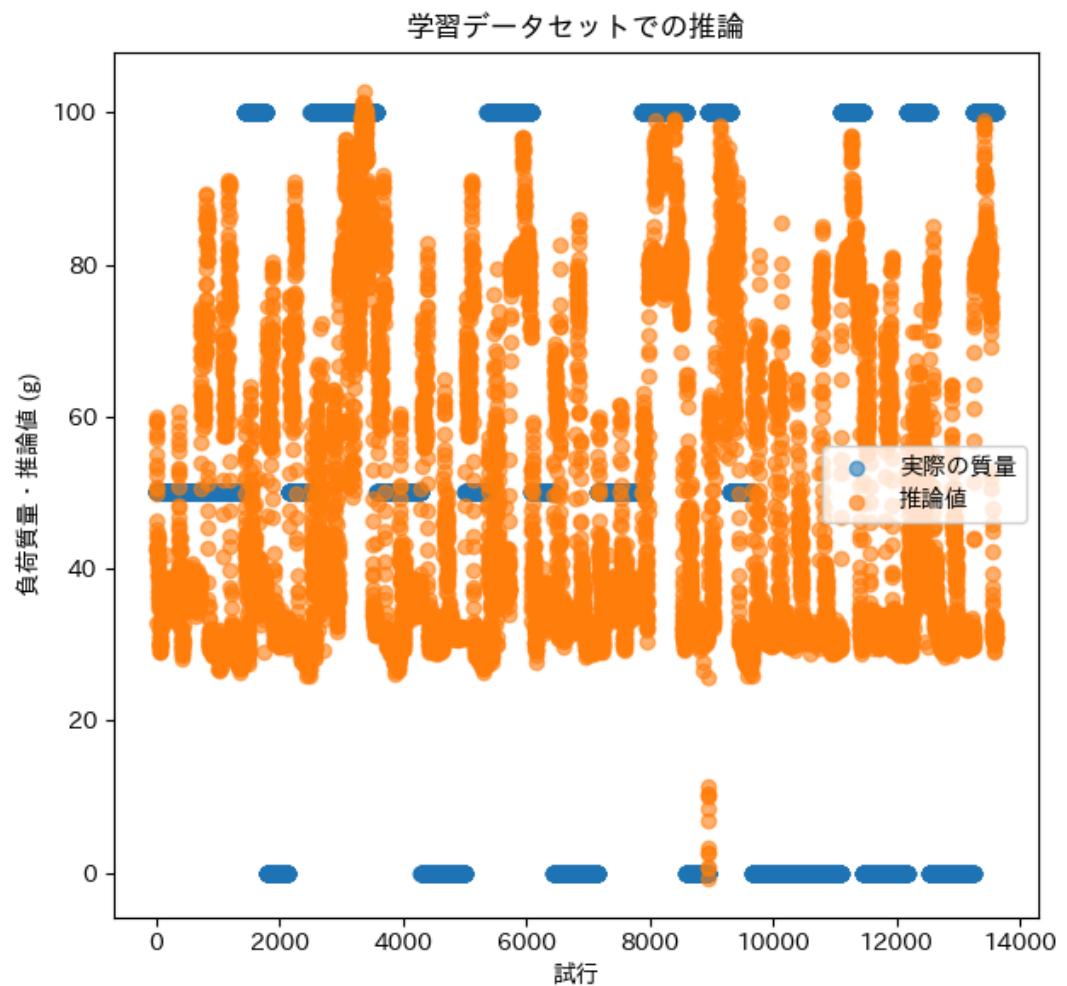


図 38: 推論モデルで学習データセットの予測をさせた結果

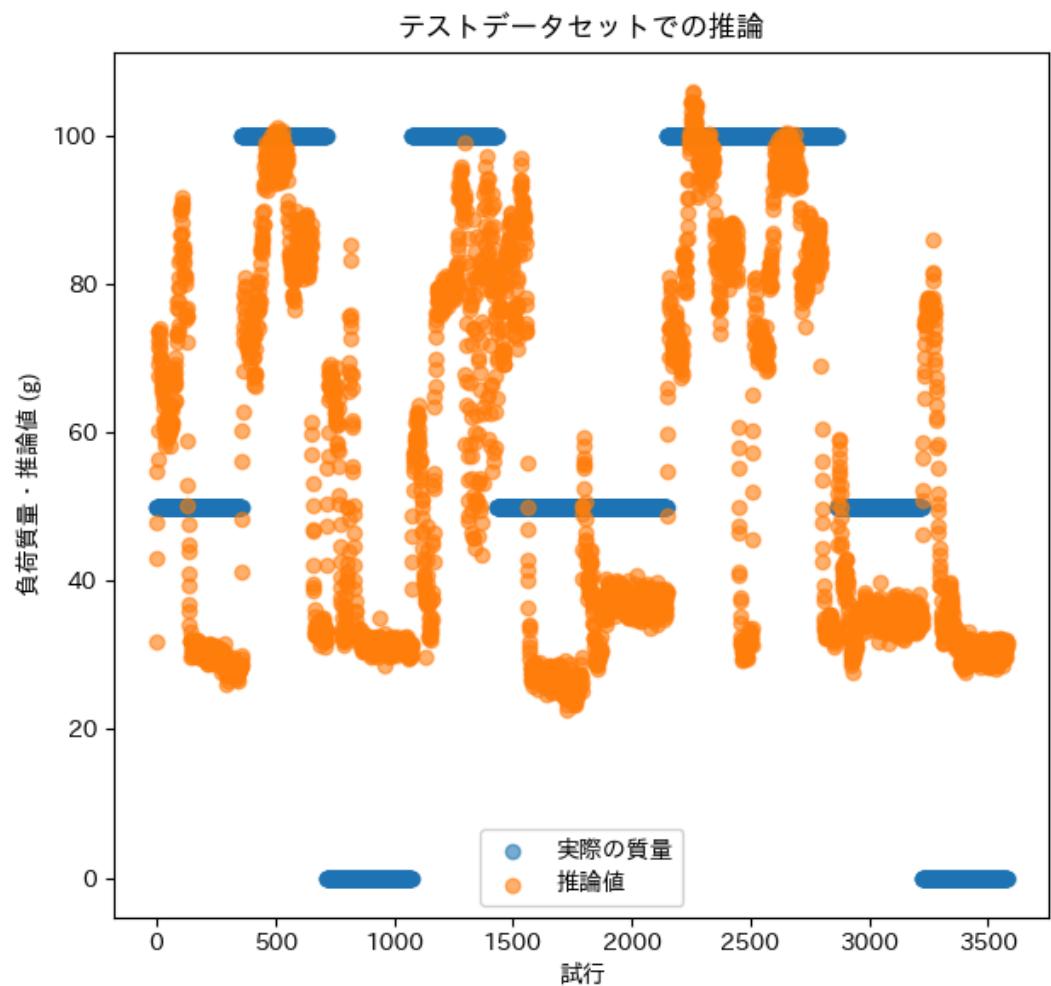


図 39: 推論モデルで評価データセットの予測をさせた結果

## 5.4 考察

学習用データを用いての評価並びに評価用データを用いての推論の RMSE はどちらも 30 を超えるものとなつた。また、両者において予測値と実際の値の差が大きいことが見て取れる。これは学習手法自体が不十分なものであることが原因と考えられる。一番の特筆すべき点としては重量なしの学習データと重量 50g、そして 100g のデータとではデータの数は同じであるにもかかわらず、推論の結果 0g 付近をほとんど予測できていない点である。これは学習手法に誤りがあつたことを示していると考えられる。

## 第6章　まとめ

筋電義手の操作性向上に必要となってくる力覚の検知を行う方法として、義手内に搭載されることの多いサーボの消費電力を用いる手法の妥当性について検討した。筋電義手におけるサーボというのは生体における筋肉のように、位置を移動させる能力を持ちながら外力が加えられた際にその大きさに対応する反応を示す。その類似性は日頃の感覚としては誰しもが気づく余地あるものではある。本研究を通してこの類似性を示すことができるか確認のための実験を重ねたが、有意な類似性を示すには至らなかつた。

しかし現時点での手法自体が間違っていると結論づけることはできない。データの与え方やリザバーコンピューティングの実装や設定の推敲を重ねることで、本来行いたかった実験の結果を得られる可能性は大いに残っていると考えられる。

## 謝辞

本研究を進めるにあたり、多大なご指導、ご助言を頂いた葛西誠也教授をはじめとする工学部情報エレクトロニクス学科量子知能デバイス研究室の構成員の皆様に深謝の意を表します。常に楽をしようと堕落の方向へ向かう私に妥協を許さず研鑽の日々を促して下さった葛西教授はもちろん、ご自身が修士論文研究や学会発表、就職活動等でお忙しいにも関わらず進んで研究内容や発表資料の校閲、的確な助言をくださった修士2年吉田聖氏、修士1年松田一希氏に深謝の意を表します。また、日頃からご指導、ご意見を頂いた、量子集積エレクトロニクス研究センター長であられる本久順一教授、先進ナノ電子材料研究室の石川史太郎教授、原真二郎准教授、量子知能デバイス研究室の佐藤威友准教授、機能通信センシング研究室の池辺将之教授、赤澤正道准教授、集積電子デバイス研究室の富岡克広准教授に深く感謝致します。

# Appendix

以下に RC で学習を行った際に用いたコードを示す。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import japanize_matplotlib
from sklearn.model_selection import train_test_split

import model

BASE_PATH: str = "data/"
file_011_name: str = "0g_1_1.csv"
file_012_name: str = "0g_1_2.csv"
file_013_name: str = "0g_1_3.csv"
file_014_name: str = "0g_1_4.csv"
file_015_name: str = "0g_1_5.csv"
file_016_name: str = "0g_1_6.csv"
file_017_name: str = "0g_1_7.csv"
file_018_name: str = "0g_1_8.csv"

file_021_name: str = "0g_2_1.csv"
file_022_name: str = "0g_2_2.csv"
file_023_name: str = "0g_2_3.csv"
file_024_name: str = "0g_2_4.csv"
file_025_name: str = "0g_2_5.csv"
file_026_name: str = "0g_2_6.csv"
file_027_name: str = "0g_2_7.csv"
file_028_name: str = "0g_2_8.csv"

file_5011_name: str = "50g_1_1.csv"
```

```

file_5012_name: str = "50g_1_2.csv"
file_5013_name: str = "50g_1_3.csv"
file_5014_name: str = "50g_1_4.csv"
file_5015_name: str = "50g_1_5.csv"
file_5016_name: str = "50g_1_6.csv"
file_5017_name: str = "50g_1_7.csv"
file_5018_name: str = "50g_1_8.csv"

file_5021_name: str = "50g_2_1.csv"
file_5022_name: str = "50g_2_2.csv"
file_5023_name: str = "50g_2_3.csv"
file_5024_name: str = "50g_2_4.csv"
file_5025_name: str = "50g_2_5.csv"
file_5026_name: str = "50g_2_6.csv"
file_5027_name: str = "50g_2_7.csv"
file_5028_name: str = "50g_2_8.csv"

file_10011_name: str = "100g_1_1.csv"
file_10012_name: str = "100g_1_2.csv"
file_10013_name: str = "100g_1_3.csv"
file_10014_name: str = "100g_1_4.csv"
file_10015_name: str = "100g_1_5.csv"
file_10016_name: str = "100g_1_6.csv"
file_10017_name: str = "100g_1_7.csv"
file_10018_name: str = "100g_1_8.csv"

file_10021_name: str = "100g_2_1.csv"
file_10022_name: str = "100g_2_2.csv"
file_10023_name: str = "100g_2_3.csv"
file_10024_name: str = "100g_2_4.csv"
file_10025_name: str = "100g_2_5.csv"
file_10026_name: str = "100g_2_6.csv"
file_10027_name: str = "100g_2_7.csv"
file_10028_name: str = "100g_2_8.csv"

def convert_data_frames_to_nparrays(
    data_frames: list[pd.DataFrame],
) -> np.ndarray[np.float64]:

```

```

# あらかじめ空のリストを作成しておく
new_data = np.empty(
    (len(data_frames), data_frames[0].columns.size,
data_frames[0].shape[0]),
    dtype=np.float64,
)

for i in range(len(data_frames)):
    for j in range(data_frames[i].columns.size):
        new_data[i][j] = data_frames[i][j].values
return new_data

def get_merged_matrix(
    list_of_files: list[str],
) -> tuple[np.ndarray[np.float64], np.ndarray[np.float64]]:
    data_frames = [
        pd.read_csv(BASE_PATH + file, header=None, skiprows=1) for file in
list_of_files
    ]
    training_frames, testing_frames = train_test_split(data_frames,
test_size=0.2)

    train_data_set = convert_data_frames_to_numpys(training_frames)
    test_data_set = convert_data_frames_to_numpys(testing_frames)
    return train_data_set, test_data_set

train_data, test_data = get_merged_matrix(
[
    file_011_name,
    file_012_name,
    file_013_name,
    file_014_name,
    file_015_name,
    file_016_name,
    file_017_name,
    file_018_name,
    file_021_name,
]

```

```
file_022_name,
file_023_name,
file_024_name,
file_025_name,
file_026_name,
file_027_name,
file_028_name,
file_5011_name,
file_5012_name,
file_5013_name,
file_5014_name,
file_5015_name,
file_5016_name,
file_5017_name,
file_5018_name,
file_5021_name,
file_5022_name,
file_5023_name,
file_5024_name,
file_5025_name,
file_5026_name,
file_5027_name,
file_5028_name,
file_10011_name,
file_10012_name,
file_10013_name,
file_10014_name,
file_10015_name,
file_10016_name,
file_10017_name,
file_10018_name,
file_10021_name,
file_10022_name,
file_10023_name,
file_10024_name,
file_10025_name,
file_10026_name,
file_10027_name,
file_10028_name,
```

```

        ]
    )

# Extract the load data from the training set. the last arrays of every set
# are the repeated load data. pick the first one of the last arrays and make
# it the load data
train_load: np.ndarray[np.float64] = np.array(
    [train_data[i][-1][0] for i in range(len(train_data))], dtype=np.float64
)
test_load: np.ndarray[np.float64] = np.array(
    [test_data[i][-1][0] for i in range(len(test_data))], dtype=np.float64
)

# erase the last arrays of every set to make the training data does not
# include load data at the end
# and transpose the data to make it compatible with the ESN model
train_data = train_data[:, :-1, :].transpose(0, 2, 1)
test_data = test_data[:, :-1, :].transpose(0, 2, 1)

# Reshape the load data to make it compatible with the RMSE calculation
# that is for example, (38,) to (38,358)
train_load_reshaped      = np.tile(train_load, (train_data.shape[1],
1)).T.reshape(-1, 1)
test_load_reshaped       = np.tile(test_load, (test_data.shape[1],
1)).T.reshape(-1, 1)

# Initialize the ESN model
input_size = train_data.shape[1] # Number of data points in each sample
reservoir_size = 100 # Size of the reservoir
output_size = 1 # Predicting a single value (Load)

esn = model.ESN(N_u=input_size, N_y=output_size, N_x=reservoir_size)

# Train the ESN model
esn.train(
    train_data,
    train_load,
    model.Tikhonov(N_x=reservoir_size, N_y=output_size, beta=1e-3),

```

```

)

# Predict using the trained model (for demonstration, use the training data
itself)
train_predictions: np.ndarray[float] = esn.predict(train_data)

# Evaluate the model by computing the mean squared error on the training
data
train_rmse: float = np.sqrt(((train_predictions - train_load_reshaped) **  

2).mean())

# Predict using the trained model
test_predictions: np.ndarray[float] = esn.predict(test_data)

# Evaluate the model by computing the mean squared error on the training
data
test_rmse: float = np.sqrt(((test_predictions - test_load_reshaped) **  

2).mean())

print("Training RMSE: ", train_rmse)

flattened_predictions = test_predictions.flatten()

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(
    x=range(len(train_load_reshaped)), y=train_load_reshaped, label="実際の  
質量", alpha=0.6
)
plt.scatter(
    x=range(len(train_predictions)), y=train_predictions, label="推論値",
alpha=0.6
)
plt.title("学習データセットでの推論")
plt.xlabel("試行")
plt.ylabel("負荷質量・推論値 (g)")
plt.legend()

plt.tight_layout()

```

```

plt.savefig(f'saved_graphs/
training_set_predictions_rmse_{train_rmse}.png')
plt.show()

print("Test RMSE: ", test_rmse)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(
    x=range(len(test_load_reshaped)), y=test_load_reshaped, label="実際の質
量", alpha=0.6
)
plt.scatter(
    x=range(len(test_predictions)), y=test_predictions, label="推論値",
alpha=0.6
)
plt.title("テストデータセットでの推論")
plt.xlabel("試行")
plt.ylabel("負荷質量・推論値 (g)")
plt.legend()

plt.tight_layout()
plt.savefig(f'saved_graphs/test_set_predictions_rmse_{test_rmse}.png')
plt.show()

```

以下にリザバーコンピューティングのモデルを示す。ただし、これは田中、中根、廣瀬（著）「リザバーコンピューティング」（森北出版）内に記載されているコードを一部改変したものであり、本ソースコードの著作権は田中剛平先生に帰属することを明記しておく。

```
import numpy as np
import networkx as nx

# 恒等写像
def identity(x):
    return x

# リザバー
class Reservoir:
    # リカレント結合重み行列Wの初期化
    def __init__(self, N_x, density, rho, activation_func, leaking_rate,
seed=0):
        """
        param N_x: リザバーのノード数
        param density: ネットワークの結合密度
        param rho: リカレント結合重み行列のスペクトル半径
        param activation_func: ノードの活性化関数
        param leaking_rate: leaky integrator モデルのリーク率
        param seed: 亂数の種
        """
        self.seed = seed
        self.W = self.make_connection(N_x, density, rho)
        self.x = np.zeros(N_x) # リザバー状態ベクトルの初期化
        self.activation_func = activation_func
        self.alpha = leaking_rate

    # リカレント結合重み行列の生成
    def make_connection(self, N_x, density, rho):
        # Erdos-Renyi ランダムグラフ
        m = int(N_x * (N_x - 1) * density / 2) # 総結合数
        G = nx.gnm_random_graph(N_x, m, self.seed)
```

```

# 行列への変換(結合構造のみ)
connection = nx.DiGraph(G)
W = np.array(connection)

# 非ゼロ要素を一様分布に従う乱数として生成
rec_scale = 1.0
np.random.seed(seed=self.seed)
W = W * np.random.uniform(-rec_scale, rec_scale, (N_x, N_x))

# スペクトル半径の計算
eigv_list = np.linalg.eig(W)[0]
sp_radius = np.max(np.abs(eigv_list))

# 指定のスペクトル半径 rho に合わせてスケーリング
W *= rho / sp_radius

return W

# リザバー状態ベクトルの更新
def __call__(self, x_in: np.ndarray):
    """
    param x_in: 更新前の状態ベクトル
    return: 更新後の状態ベクトル
    """
    # self.x = self.x.reshape(-1, 1)
    self.x = (1.0 - self.alpha) * self.x + self.alpha * \
self.activation_func(
        np.dot(self.W, self.x)
        + x_in # self.x は reservoir のノード数 (N_x) の次元を持つベクトル
    )
    return self.x

# リザバー状態ベクトルの初期化
def reset_reservoir_state(self):
    self.x *= 0.0

# 出力層

```

```

class Output:
    # 出力結合重み行列の初期化
    def __init__(self, N_x, N_y, seed=0):
        """
        param N_x: リザバーのノード数
        param N_y: 出力次元
        param seed: 乱数の種
        """
        # 正規分布に従う乱数
        np.random.seed(seed)
        self.Wout = np.random.normal(size=(N_y, N_x))

    # 出力結合重み行列による重みづけ
    def __call__(self, x):
        """
        param x: N_x 次元のベクトル
        return: N_y 次元のベクトル
        """
        return np.dot(self.Wout, x)

    # 学習済みの出力結合重み行列を設定
    def setweight(self, Wout_opt):
        self.Wout = Wout_opt

# 出力フィードバック
class Feedback:
    # フィードバック結合重み行列の初期化
    def __init__(self, N_y, N_x, fb_scale, seed=0):
        """
        param N_y: 出力次元
        param N_x: リザバーのノード数
        param fb_scale: フィードバックスケーリング
        param seed: 乱数の種
        """
        # 一様分布に従う乱数
        np.random.seed(seed)
        self.Wfb = np.random.uniform(-fb_scale, fb_scale, (N_x, N_y))

```

```

# フィードバック結合重み行列による重みづけ
def __call__(self, y):
    """
    param y: N_y 次元のベクトル
    return: N_x 次元のベクトル
    """
    return np.dot(self.Wfb, y)

# リッジ回帰 (beta=0 のときは線形回帰)
class Tikhonov:
    def __init__(self, N_x, N_y, beta):
        """
        param N_x: リザバーのノード数
        param N_y: 出力次元
        param beta: 正則化パラメータ
        """
        self.beta = beta
        self.X_XT = np.zeros((N_x, N_x))
        self.D_XT = np.zeros((N_y, N_x))
        self.N_x = N_x

    # 学習用の行列の更新
    def __call__(self, d, x):
        d = np.reshape(d, (-1, 1))
        x = np.reshape(x, (-1, 1))
        self.X_XT = self.X_XT + np.dot(x, x.T)
        self.D_XT = self.D_XT + np.dot(d, x.T)

    # Wout の最適解（近似解）の導出
    def get_Wout_opt(self):
        X_pseudo_inv = np.linalg.inv(self.X_XT + self.beta * 
np.identity(self.N_x))
        Wout_opt = np.dot(self.D_XT, X_pseudo_inv)
        return Wout_opt

# 逐次最小二乗 (RLS) 法
class RLS:

```

```

def __init__(self, N_x, N_y, delta, lam, update):
    """
    param N_x: リザバーのノード数
    param N_y: 出力次元
    param delta: 行列 P の初期条件の係数 (P=delta*I, 0<delta<<1)
    param lam: 忘却係数 (0<lam<1, 1に近い値)
    param update: 各時刻での更新繰り返し回数
    """

    self.delta = delta
    self.lam = lam
    self.update = update
    self.P = (1.0 / self.delta) * np.eye(N_x, N_x)
    self.Wout = np.zeros([N_y, N_x])

# Wout の更新
def __call__(self, d, x):
    x = np.reshape(x, (-1, 1))
    for i in np.arange(self.update):
        v = d - np.dot(self.Wout, x)
        gain = 1 / self.lam * np.dot(self.P, x)
        gain = gain / (1 + 1 / self.lam * np.dot(np.dot(x.T, self.P),
x))
        self.P = 1 / self.lam * (self.P - np.dot(np.dot(gain, x.T),
self.P))
        self.Wout += np.dot(v, gain.T)

    return self.Wout

# エコーステートネットワーク
class ESN:
    # 各層の初期化
    def __init__(
        self,
        N_u,
        N_y,
        N_x,
        density=0.05,
        input_scale=1.0,

```

```

    rho=0.95,
    activation_func=np.tanh,
    fb_scale=None,
    fb_seed=0,
    noise_level=None,
    leaking_rate=1.0,
    output_func=identity,
    inv_output_func=identity,
    classification=False,
    average_window=None,
):
    """
    param N_u: 入力次元
    param N_y: 出力次元
    param N_x: リザバーのノード数
    param density: リザバーのネットワーク結合密度
    param input_scale: 入力スケーリング
    param rho: リカレント結合重み行列のスペクトル半径
    param activation_func: リザバーノードの活性化関数
    param fb_scale: フィードバックスケーリング (default: None)
    param fb_seed: フィードバック結合重み行列生成に使う乱数の種
    param leaking_rate: leaky integrator モデルのリーク率
    param output_func: 出力層の非線形関数 (default: 恒等写像)
    param inv_output_func: output_func の逆関数
    param classification: 分類問題の場合は True (default: False)
    param average_window: 分類問題で出力平均する窓幅 (default: None)
    """

    # self.Input = Input(N_u, N_x, input_scale)
    self.Reservoir = Reservoir(N_x, density, rho, activation_func,
leaking_rate)
    self.Output = Output(N_x, N_y)
    self.N_u = N_u
    self.N_y = N_y
    self.N_x = N_x
    self.y_prev = np.zeros(N_y)
    self.output_func = output_func
    self.inv_output_func = inv_output_func
    self.classification = classification

```

```

# 出力層からのリザバーへのフィードバックの有無
if fb_scale is None:
    self.Feedback = None
else:
    self.Feedback = Feedback(N_y, N_x, fb_scale, fb_seed)

# リザバーの状態更新におけるノイズの有無
if noise_level is None:
    self.noise = None
else:
    np.random.seed(seed=0)
    self.noise = np.random.uniform(-noise_level, noise_level,
(self.N_x, 1))

# 分類問題か否か
if classification:
    if average_window is None:
        raise ValueError("Window for time average is not given!")
    else:
        self.window = np.zeros((average_window, N_x))

# バッチ学習
def train(self, U, D, optimizer, trans_len=None):
    """
    U: 教師データの入力, データ長×N_u
    D: 教師データの出力, データ長×N_y
    optimizer: 学習器
    trans_len: 過渡期の長さ
    return: 学習前のモデル出力, データ長×N_y
    """

    train_len = len(U)
    if trans_len is None:
        trans_len = 0
    Y = []

    # 時間発展
    for n in range(train_len):
        for m in range(len(U[1])):  # データ数だけループさせる
            # x_in = self.Input(U[n])

```

```

# TODO: x_in の大きさ > リザバーのノード数(N_x) の場合の処理を追
加する
x_in = np.zeros(self.N_x)
x_temp = U[n][m]
x_in[: len(U[n][m])] = x_temp

# フィードバック結合
if self.Feedback is not None:
    x_back = self.Feedback(self.y_prev)
    x_in = x_in + x_back

# ノイズ
if self.noise is not None:
    x_in += self.noise

# リザバー状態ベクトル
x = self.Reservoir(x_in)

# 分類問題の場合は窓幅分の平均を取得
if self.classification:
    self.window = np.append(self.window, x.reshape(1, -1),
axis=0)
    self.window = np.delete(self.window, 0, 0)
    x = np.average(self.window, axis=0)

# 目標値
d = D[n]
d = self.inv_output_func(d)

# 学習器
if n > trans_len: # 過渡期を過ぎたら
    optimizer(d, x)

# 学習前のモデル出力
y = self.Output(x)
Y.append(self.output_func(y))
self.y_prev = d

# 学習済みの出力結合重み行列を設定

```

```

    self.Output.setweight(optimizer.get_Wout_opt())

    # モデル出力（学習前）
    return np.array(Y)

# バッチ学習後の予測
def predict(self, U):
    test_len = len(U)
    Y_pred = []

    # 時間発展
    for n in range(test_len):
        for m in range(len(U[1])):  # データ数だけループさせる
            # x_in = self.Input(U[n])
            # TODO: x_in の大きさ > リザバーのノード数(N_x) の場合の処理を追
            加する
            x_in = np.zeros(self.N_x)
            x_temp = U[n][m]
            x_in[:len(U[n][m])] = x_temp

            # フィードバック結合
            if self.Feedback is not None:
                x_back = self.Feedback(self.y_prev)
                x_in += x_back

            # リザバー状態ベクトル
            x = self.Reservoir(x_in)

            # 分類問題の場合は窓幅分の平均を取得
            if self.classification:
                self.window = np.append(self.window, x.reshape(1, -1),
axis=0)
                self.window = np.delete(self.window, 0, 0)
                x = np.average(self.window, axis=0)

            # 学習後のモデル出力
            y_pred = self.Output(x)
            Y_pred.append(self.output_func(y_pred))
            self.y_prev = y_pred

```

```
# モデル出力（学習後）
return np.array(Y_pred)
```

## 参考文献

- [1] V. Mendez, F. Iberite, S. Shokur, and S. Micera, “Current Solutions and Future Trends for Robotic Prosthetic Hands.” Annual Review of Control, Robotics, and Autonomous Systems, 2021. [Online]. Available: <https://doi.org/10.1146/annurev-control-071020-104336>
- [2] “【2024 年版】力覚センサー メーカー 13 社一覧.” [Online]. Available: <https://metoree.com/categories/force-sensor/>
- [3] 齊藤佑貴, “センサレスセンシング,” in ハプティクスとその応用, シーエムシー出版, 2022.
- [4] 田中剛平, 中根了昌, and 廣瀬明, リザバーコンピューティング. 森北出版.
- [5] Y. Yada, S. Yasuda, and H. Takahashi, “Physical reservoir computing with FORCE learning in a living neuronal culture,” vol. 119. Applied Physics Letters.
- [6] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, “Improving reservoirs using intrinsic plasticity,” vol. 71. Neurocomputing. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231208000519>
- [7] “Ridge Regression.” [Online]. Available: <https://online.stat.psu.edu/stat857/node/155/>