—修士学位論文—

機械学習を用いた複数環境に対する適応的力制御器の設計に関する研究

Design of Adaptive Force Controller for Multiple Environtments based on Machine Learning

令和元年度

三重大学大学院 工学研究科
博士前期課程 電気電子工学専攻
辻井 祥太郎

目次

第 1 章　緒言　1

1.1　研究背景と目的 1

1.2　論文の構成 3

第 2 章　データ駆動型制御器設計法と機械学習　5

2.1　データ駆動制御器設計法 5

2.1.1　モデル参照制御問題 5

2.1.2　Virtual Reference Feedback Tuning (VRFT) 6

2.1.3　VRFT におけるプレフィルタの選択 8

2.2　クラスタリング 10

2.2.1　階層型クラスタリング 10

2.3　Support Vector Machine 11

2.3.1　識別関数とマージン最小化 12

2.3.2　カーネルトリックによる非線形識別 13

2.3.3　SVM パラメータの探索 14

2.3.4　マルチクラス SVM 16

第 3 章　切替型制御器の設計法　17

3.1　切替型制御器の数学的表現 17

3.2　切替時の制御器状態量補償 18

3.3　切替時のチャタリングの防止 19

3.4　切替型制御器の設計方法 20

3.5　一般的な制御対象への拡張 22

第 4 章　ゲインスケジュールド制御器の設計　23

4.1　ゲインスケジュールド制御系の数学的表現 23

4.1.1 切替型制御系の問題点 23
4.2 ゲインスケジュールド制御器 23
4.3 スケジューリング則 24

第 5 章 力制御系のモデリング **26**
5.1 制御対象のモデリング 26
5.2 外乱オブザーバ 28
5.3 ノミナルプラントモデル 28

第 6 章 シミュレーションによる検証 **30**
6.1 シミュレーション条件と設計条件 30
6.2 候補制御器の設計 32
6.2.1 データの分類 32
6.2.2 VRFT による候補制御器設計 33
6.3 適応的制御器の時間応答 36

第 7 章 実機実験 **40**
7.1 実機システム 40
7.1.1 インピーダンス制御 40
7.2 実験条件 42
7.2.1 システムの周波数特性の確認 43
7.3 候補制御器の設計結果 45
7.4 適応的制御器の時間応答 48

第 8 章 結言 **51**
8.1 まとめ 51
8.2 今後の課題 52

参考文献 **55**

謝辞 **58**

論文目録 **59**

付録 A 実機システムについて **60**

A.1 ハードウェア仕様 60
A.1.1 リニアモータ 61
A.1.2 サーボドライバ 61
A.1.3 制御盤とスイッチボックス 64
A.1.4 構造可変ロボットと共用する部分 66
A.2 ソフトウェア仕様 68
A.2.1 リアルタイム OS 68
A.2.2 サーボドライバとの通信方法 72
A.3 実験手順 73

第1章　緒言

1.1　研究背景と目的

近年，協働型ロボットや知能型ロボットは実用的な応用が進んでいる[1]。これらのロボットでは人間や環境と接触が考慮されており，従来不可能だった作業が可能となることが期待されている。ロボットにおいて接触を考慮するためには力制御が有効であるため，接触対象との間に生じる力を制御する力制御の重要性はますます増している。力制御は間接力制御と直接力制御の 2 つに分類される。間接力制御は外部からの力とロボットの位置との間で所望の関係を満たすように運動を制御するものである。この方法で代表的な手法はインピーダンス制御[2] であり，所望のインピーダンス特性が成り立つように位置制御を行う。間接的な手法は人との安全な接触を実現可能であるため，協働型ロボットにおける応用例[3] が報告されている。しかし，直接型力制御では精密な力制御が可能であるため，切削や研磨などの作業を自動化するためには直接型力制御の方がより有用であると考えられる。直接型力制御の代表例は位置と力のハイブリッド制御[4] である。これは，作業空間上で力制御を行う軸とそれ以外の位置制御を行う軸に分けてそれぞれ制御する手法である。実際にこの方法で仕上げ加工を自動化し，従来よりも高い精度で作業が可能であるという例が報告されている[5]。よって，本研究では直接型力制御に注目する。

力制御における問題の一つに，不確かさを持つ環境に対して性能や安定性を維持することがある。力制御系は閉ループ内に環境の動特性を含むため，もし環境が変化した場合にはシステム全体の性能や安定性は悪化する可能性がある。この問題に対して，環境変動を不確かさとして抑え込むロバスト制御を用いた手法と環境変化に合わせて変化する時変の制御器を用いる手法が存在する。前者では，例えば，2 自由度ロバストサーボ系を用いた力制御が提案されている[6]。しかし，環境が変化する前提で性能を直接的に向上するためには，非線形な適応制御器を用いるのが自然である。

非線形な適応制御器の設計法についてもこれまで様々提案されてきた。例えば力制御の文脈においては，パラメータ推定に基づく適応制御[7]，ニューラルネットワークを制御器に用いる方法[8]，ファジィ制御に基づく方法[9] などが考案されてきた。しかし，これ

らの多くは未知環境への適応性を持つものの，環境が変化するごとに推定時間や学習時間が必要である。そのため適応する前に性能劣化や不安定化する恐れがあった。そこで，そのような遅れによる問題を避けるために，オンライン調整のみに頼るのではなく，事前情報を用いたオフライン設計とオンライン調整の混合的な手法に注目する。

混合的な設計手法には，一般的な線形制御器の設計と同様にモデルベースなアプローチとデータ駆動アプローチがある。前者のモデルベース制御器設計では制御対象の数理モデルを用いて設計を行うため，数学的取り扱いが容易である利点がある。その一方で，モデルを作成するためのシステム同定に手間がかかる上，設計した制御器を実装した時所望の特性を実現できない場合には再度同定実験をやり直す必要がある。つまり，システムの次数や複雑さなどシステム同定の精度が制御器設計へ大きく影響してしまう。後者のデータ駆動型制御器設計では制御対象から得た時系列データを直接用いて設計を行うため，システム同定の手順なしに制御器を直接設計できる。そのため，システム同定の精度への依存せず，制御性能を陽に評価して試行錯誤なしに設計が可能である。これらの考え方を力制御に適用すると，モデルベースの場合は制御対象を Piecewise Affine (PWA) システム[10], [11] や Linear Parameter Varying (LPV) システム[12] として同定することとなる。しかし，所望の制御性能を達成するために必要最低限な複雑さと正確さを持つモデルを同定するまでには試行錯誤が伴う。これに対して，データ駆動型の場合では，所望の制御性能を満たすために必要な制御器の複雑さを直接評価して設計可能である。データ駆動型に基づいた混合的な設計手法は，非線形システムに対する Virtual Reference Feedback Tuning (VRFT)[13]，RBFN を用いた非線形 VRFT[14]，機械学習に基づいた切替型制御器[15] や混合感度問題に基づいた切替型制御器[16]，データベース駆動型制御器 (DD-PID)[17] などが考案されている。本研究では，文献 [15] を力制御系へ拡張することを考える。

文献 [15] では，線形な VRFT[18] とクラスタリングに基づいた PWA システムの同定法[19] を用いた切替型制御器の設計法が提案されている。切替型制御器は，複数の線形制御器とそれらを切り替えるスーパバイザから構成される[20]。この方法では，クラスタリングを用いて制御対象の時系列データを線形システムごとに分類し，分類されたデータから線形制御器を VRFT でオフライン設計している。また，Support Vector Machine (SVM) を用いて事前に時系列データの識別器を用意し，これをスーパーバイザとして用いる。そして，オンラインでは適切な線形制御器への切替のみをを行う。これにより，1 つの状態量のみで非線形性が変化する対象に対して，制御対象の非線形性を補償することが可能であることが報告されている。しかし，より一般には複数の状態量の間の関係に

よって非線形性が変化する場合がある。例えば，力制御の場合には反力と位置及び速度の間の関係性によって非線形性が決定される。

そこで本研究では，文献 [15] の切替型力制御器の設計法をより一般的な場合に拡張し，環境変動に対応可能な制御器の設計法を提案することを目的とする。複数の状態量から非線形性が決定される場合には，従来手法よりも非線形性の識別が困難となる。そのため，SVM のカーネル関数を線形カーネルから Radial Basis Funtion (RBF) カーネルへ変更しより識別能力を向上させる。また，切替型制御器では切替時に制御系を乱さないために状態量補償[21], [22] が必要となる。しかし，データ駆動型のフレームワークにおいては制御対象の零点の情報が利用できないため，状態量の補償は保守的となってしまう。さらに，もし事前に線形制御器を設計しておいた環境以外の中間の特性を持つ環境に接触した場合には，切替型制御器は中間的な特性を取ることはできない。そこで，切替自体を行うことなく，切替型制御器を内挿するように可変ゲインを滑らかに変化させるゲインスケジュールド制御器へ拡張する。これによって，複数の環境が混合した時系列データから適応的な制御器を設計することができる。そして，提案手法はリニアモータを用いた 1 自由度ロボットに対するシミュレーションと実機実験において検証する。

1.2　論文の構成

本研究では，データ駆動制御器設計法と機械学習を利用した適応的制御器の設計法の提案を行い，提案手法を複数環境に対する力制御系に適用する。

本論文は全 6 章で構成されており，各章の構成は以下のとおりである。

第 2 章　データ駆動型制御器設計法と機械学習

提案手法の構成要素となる手法について説明する。初めに提案手法で利用する線形システムに対する VRFT[18] について述べる。そして，データ分類と識別に用いるクラスタリング[23] との SVM[24] について説明する。

第 3 章　切替型制御器の設計

文献 [15] に基づいて，前章で説明した要素手法を用いた切替型制御器の設計法を説明する。そして，より一般のシステムへ拡張方法について説明する。

第 4 章　ゲインスケジュールド制御器の設計

切替型制御器を拡張し，ゲインスケジュールド制御器の設計法を提案する。

第 5 章　力制御系のモデリング

本研究で対象とする 1 自由度ロボットによる直接力制御系のモデリングを行う。

そして，環境が変化する場合には複数の状態量で決まる非線形特性を持つことを示す。

第 6 章　シミュレーション

力制御系のシミュレーションによって，提案手法の制御器の有効性を検証する。

第 7 章　実機実験

2 つのスライダを持つリニアモータを用いた実機実験によって，シミュレーションと同様に提案手法が有効であることを検証する。

第 8 章　まとめと今後の課題

本研究のまとめと今後の課題について述べる。

さらに，実機実験に際しては，力センサを介して力を及ぼすような 2 つのスライダを持つリニアモータシステムを作成したため，付録にて概要を記す。

第2章　データ駆動型制御器設計法と機械学習

本章では，本研究で提案する設計法において用いる要素手法について説明する。初めにデータ駆動型制御器設計法の 1 つである線形 VRFT について説明する。そして，機械学習の手法であるクラスタリングと SVM についてそれぞれ説明する。

2.1　データ駆動制御器設計法

本節では，データ駆動制御器設計法の 1 つである VRFT について説明していく。本研究では，この VRFT を利用することで適応的制御器を構成するそれぞれの区分的線形制御器の設計を行う。まず，モデルベース制御器設計法におけるモデル参照制御問題について述べる。その後，VRFT のアルゴリズムについて説明する。

2.1.1　モデル参照制御問題

モデル参照制御問題とは，設計者が任意に決定できる参照モデル $M(z)$ に対して，制御対象 $P(z)$ と制御器 $C(z,\boldsymbol{\rho})$ で形成した閉ループシステムとの特性の差が最小になる制御器 $C(z,\boldsymbol{\rho})$ を設計する問題である。これは，図 2.1 のように表すことができる。モデル参照制御問題は，参照モデルと閉ループシステムとの誤差システムの 2 ノルムの 2 乗を最小化する $\boldsymbol{\rho}$ を求める問題として (2.1) 式のように定式化できる。

$$\begin{aligned} \boldsymbol{\rho}_{\mathrm{MR}} &= \arg\min_{\boldsymbol{\rho}} \ J_{\mathrm{MR}}(\boldsymbol{\rho}) \\ J_{\mathrm{MR}}(\boldsymbol{\rho}) &= \left\| \left(M(z) - \frac{P(z)C(z,\boldsymbol{\rho})}{1+P(z)C(z,\boldsymbol{\rho})} \right) W(z) \right\|_2^2 \end{aligned} \tag{2.1}$$

ここで，$W(z)$ は設計者が与える周波数重みである。(2.1) 式の最小化は制御対象のモデル $P(z)$ を用いることができる場合には H_2 制御問題となり，容易に $C(z,\boldsymbol{\rho})$ を求めることができる。しかし，データ駆動型制御器設計法では制御対象のモデル $P(z)$ を用いないため，制御対象の入出力データを用いて近似的に (2.1) 式を評価する問題を考える。

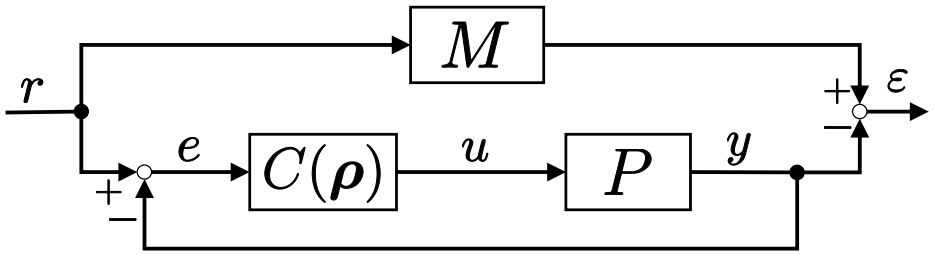


図 2.1: モデル参照制御問題

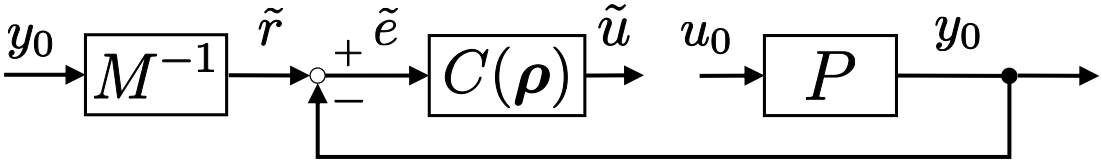


図 2.2: VRFT の基本的なアイデア

2.1.2　Virtual Reference Feedback Tuning (VRFT)

本節では，データ駆動型制御器設計法の 1 つである Virutal Reference Feedback Tuning (VRFT)[18] のアルゴリズムについて説明する。初めに VRFT の基本的なアイデアを図 2.2 に示す。VRFT では，図 2.2 のように参照モデル $M(z)$ を用いて仮想参照信号 $\tilde{r}(t)$ を生成する。ここで，仮想的な制御器の出力 $\tilde{u}(t)$ が実際にデータ取得した際の入力データ $u_0(t)$ が一致するならば，参照モデルと閉ループシステムの応答が一致することが分かる。以上の考えから，$\tilde{u}(t)$ と $u_0(t)$ が近づくような制御器を求めるのが VRFT である。

ここからは，VRFT の最適化問題の定式化を行う。制御器 $C(z,\boldsymbol{\rho})$ をパラメータ $\boldsymbol{\rho}$ と制御器の構造を表す伝達関数 $\boldsymbol{\beta}(z)$ を用いて以下のように定義する。

$$C(z,\boldsymbol{\rho}) = \boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{\beta}(z) \tag{2.2}$$

$$\boldsymbol{\rho} = [\rho_1, \rho_2, \ldots, \rho_{n_b}]^{\mathrm{T}} \tag{2.3}$$

$$\boldsymbol{\beta}(z) = [\beta_1(z), \beta_2(z), \ldots, \beta_{n_b}(z)]^{\mathrm{T}} \tag{2.4}$$

ここで，n_b は制御器パラメータの数である。

この時，VRFT では以下のアルゴリズムで設計を行う。まず，図 2.3 に示すような開ループ実験を行い，設計者が用意した入力 $u_0(t)$ を制御対象 $P(z)$ に印加した時の出力 $y_0(t)$ を取得する。ここで，参照信号 $r(t)$ から出力 $y(t)$ までの設計者が実現したい閉ループ特性を参照モデル $M(z)$ として与える。この時，(2.5) 式を満たすような仮想参照信号

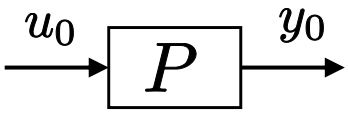


図 2.3: 入出力データの取得

$\tilde{r}(t)$ を逆算することができる。

$$y_0(t) = M(z)\tilde{r}(t) \tag{2.5}$$

この仮想参照信号 $\tilde{r}(t)$ を用いることで，$\tilde{r}(t)$ を閉ループに入力したときの制御器の出力 $\tilde{u}(t, \boldsymbol{\rho})$ は (2.6) 式で仮想的に計算できる。

$$\begin{aligned} \tilde{u}(t, \boldsymbol{\rho}) &= C(z, \boldsymbol{\rho})(\tilde{r}(t) - y_0(t)) \\ &= C(z, \boldsymbol{\rho})(M(z)^{-1} - 1)y_0(t) \end{aligned} \tag{2.6}$$

ただし，$M(z)^{-1}$ がプロパーとは限らないため，プロパー性を保つためのプレフィルタが必要となる。文献 [18] より，(2.7) 式を満たすプレフィルタ $L(z)$ を導入する。

$$|L(z)|^2 = |1 - M(z)|^2 |M(z)|^2 |W(z)|^2 \frac{1}{\phi_u(\omega)} \tag{2.7}$$

ここで，$W(z)$ は設計者が与える周波数重み関数，$\phi_u(\omega)$ は $u_0(t)$ のスペクトル密度を表す。プレフィルタの選択については 2.1.3 節で述べる。このプレフィルタ $L(z)$ を通した制御入力 $u_L(t)$ と仮想制御入力 $\tilde{u}_L(t)$ は以下で表せる。

$$u_L(t) = L(z)u_0(t) \tag{2.8}$$

$$\begin{aligned} \tilde{u}_L(t, \boldsymbol{\rho}) &= L(z)\tilde{u}(t, \boldsymbol{\rho}) \\ &= C(z, \boldsymbol{\rho})(M(z)^{-1} - 1)y_L(t) \end{aligned} \tag{2.9}$$

ここで，$y_L(t) = L(z)y_0(t)$ を用いた。

前述の基本的なアイデアより，$\tilde{u}(t)$ と $u_0(t)$ が近づけば，参照モデル $M(z)$ と実際の閉ループ特性 $\frac{C(z,\boldsymbol{\rho})P(z)}{1+C(z,\boldsymbol{\rho})P(z)}$ は近づく。従って，この条件を満たすような制御器パラメータ $\boldsymbol{\rho}$ を導出することで所望の特性を実現する制御器を設計することができる。よって，入出力データを N サンプル取得したとき，$u_L(t) - \tilde{u}_L(t, \boldsymbol{\rho})$ を最小化する制御器パラメータ $\overline{\boldsymbol{\rho}}_N$ を求める問題は (2.10) 式のように定式化できる。

$$\begin{aligned} \overline{\boldsymbol{\rho}}_N &= \arg\min_{\boldsymbol{\rho}} \ J^N_{\mathrm{VR}}(\boldsymbol{\rho}) \\ J^N_{\mathrm{VR}}(\boldsymbol{\rho}) &= \frac{1}{N}\sum_{k=1}^{N} \left(u_L(t) - \tilde{u}_L(t, \boldsymbol{\rho})\right)^2 \end{aligned} \tag{2.10}$$

さらに，制御器は (2.2) 式のように線形に定義されているため，(2.10) 式の問題は最小二乗法に帰着させることができる。まず，(2.2) 式と (2.9) 式より，制御器内部の信号 $\boldsymbol{\varphi}(t)$ を (2.11) 式とする。

$$\begin{aligned}\boldsymbol{\varphi}(t) &= \boldsymbol{\beta}(z)\left(M(z)^{-1}-1\right)\ y_L(t)\\ &= \boldsymbol{\beta}(z)\ \tilde{e}_L\end{aligned} \tag{2.11}$$

ただし，$\tilde{e}_L = \left(M(z)^{-1}-1\right)\ y_L(t)$ とする。この (2.11) 式を利用することで，(2.10) 式は (2.12) 式と書き直せる。

$$\begin{aligned}\overline{\boldsymbol{\rho}}_N &= \arg\min_{\boldsymbol{\rho}} J_{\mathrm{VR}}^N(\boldsymbol{\rho})\\ J_{\mathrm{VR}}^N(\boldsymbol{\rho}) &= \frac{1}{N}\sum_{k=1}^{N}(u_L(t)-\boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{\varphi}(t))^2\end{aligned} \tag{2.12}$$

この (2.12) 式はパラメータに対して線形な形で記述されているため，最小二乗法を用いて最適なパラメータを導出できる。以上より，(2.12) 式を最小化するパラメータ $\overline{\boldsymbol{\rho}}_N$ は最小二乗法より (2.13) 式で求めることができる。

$$\overline{\boldsymbol{\rho}}_N = \left[\sum_{k=1}^{N}\boldsymbol{\varphi}(t)\boldsymbol{\varphi}(t)^{\mathrm{T}}\right]^{-1}\sum_{k=1}^{N}\boldsymbol{\varphi}(t)u_L(t) \tag{2.13}$$

これにより，制御対象のモデル $P(z)$ を用いることなく，入出力時系列データのみを用いて制御器設計を行うことができる。

2.1.3　VRFT におけるプレフィルタの選択

本節では，(2.8) 式と (2.9) 式で導入したプレフィルタ $L(z)$ について説明する。このプレフィルタ $L(z)$ は，本来最小化すべきモデルベース制御器設計法の評価関数 $J_{\mathrm{MR}}(\boldsymbol{\rho})$ と VRFT の評価関数 $J_{\mathrm{VR}}(\boldsymbol{\rho})$ の意味を一致させることと，仮想制御入力 $\tilde{u}_L(t)$ を計算する際のプロパ性を保つことを目的に導入する。(2.1) 式のモデル参照制御問題における評価関数 $J_{\mathrm{MR}}(\boldsymbol{\rho})$ は，誤差システムの 2 ノルムを評価していることから，(2.14) 式のように書き換えることができる。

$$\begin{aligned}J_{\mathrm{MR}}(\boldsymbol{\rho}) &= \left\|\left(M(z)-\frac{P(z)C(z,\boldsymbol{\rho})}{1+P(z)C(z,\boldsymbol{\rho})}\right)W(z)\right\|_2^2\\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\frac{|P(\omega)|^2|W(\omega)|^2}{|1+P(\omega)C^*(\omega)|^2}\frac{|C^*(\omega)-C(\omega,\boldsymbol{\rho})|^2}{|1+P(\omega)C(\omega,\boldsymbol{\rho})|^2}d\omega\end{aligned} \tag{2.14}$$

ここで，$C^*(z)$ は (2.15) 式を満たす理想制御器である。

$$M(z) = \frac{P(z)C^*(z)}{1 + P(z)C^*(z)} \tag{2.15}$$

また，VRFT の評価関数 $J_{\mathrm{VR}}^N(\boldsymbol{\rho})$ において入出力データ数が十分多い，つまり $N \to \infty$ としたとき，(2.16) 式の関係が得られる。

$$\lim_{N\to\infty} J_{\mathrm{VR}}^N = J_{\mathrm{VR}} = E[(u_L(t) - C(z,\boldsymbol{\rho})\tilde{e}_L(t))^2] \tag{2.16}$$

ここで，$E[\bullet]$ は期待値を表す。さらに，これを前述の理想制御器 $C^*(z)$ を用いて (2.17) のように書き換える。

$$\begin{aligned}
J_{\mathrm{VR}} &= E\left[(u_L(t) - C(z,\boldsymbol{\rho})\tilde{e}_L(t))^2\right] \\
&= E\left[L^2(z)\left\{u_0(t) - C(z,\boldsymbol{\rho})\left(M^{-1}(z) - 1\right)y_0(t)\right\}^2\right] \\
&= E\left[L^2(z)\left\{u_0(t) - C(z,\boldsymbol{\rho})\left(\frac{1-M(z)}{M(z)}\right)P(z)u_0(t)\right\}^2\right] \\
&= E\left[L^2(z)\left\{\left(\frac{1-M(z)}{M(z)}\right)C^*(z)P(z) - C(z,\boldsymbol{\rho})P(z)\left(\frac{1-M(z)}{M(z)}\right)\right\}^2 {u_0}^2(t)\right] \\
&= E\left[L^2(z)P^2(z)\left(\frac{1-M(z)}{M(z)}\right)^2 \left(C^*(z) - C(z,\boldsymbol{\rho})\right)^2 {u_0}^2(t)\right]
\end{aligned} \tag{2.17}$$

時間領域の信号のエネルギーは周波数領域の信号のエネルギーを評価するのと等しいというパーセバルの定理を用いると，(2.17) 式は (2.18) 式へ書き換えることができる。

$$J_{\mathrm{VR}} = \frac{1}{2\pi}\int_{-\pi}^{\pi} |P(\omega)|^2 |C^*(\omega) - C(\omega,\boldsymbol{\rho})|^2 \frac{|1-M(\omega)|^2}{|M(\omega)|^2} |L|^2 \Phi_u(\omega) d\omega \tag{2.18}$$

ここで，$\Phi_u(\omega)$ は $u_0(t)$ のパワースペクトル密度である。(2.14) 式と (2.18) 式を比較すると，プレフィルタ L を (2.19) 式とすることで $J_{\mathrm{MR}}(\boldsymbol{\rho}) = J_{\mathrm{VR}}(\boldsymbol{\rho})$ となり，$J_{\mathrm{MR}}(\boldsymbol{\rho})$ を最小化するパラメータと $J_{\mathrm{VR}}(\boldsymbol{\rho})$ を最小化するパラメータが一致する。

$$|L(z)| = \frac{|M(z)|^2|W(z)|^2}{|1 + P(z)C(z,\boldsymbol{\rho})|^2}\frac{1}{\Phi_u} \tag{2.19}$$

しかし，(2.19) 式は制御対象のモデル $P(z)$ や制御器パラメータ $\boldsymbol{\rho}$ に依存している。そこで，最適化後の制御器は十分に理想制御器に近いこと，つまり $C(z,\boldsymbol{\rho}) = C^*(z)$ と仮定する。これにより，(2.7) 式のように既知のパラメータのみでプレフィルタを記述できる。

$$|L(z)|^2 = |1 - M(z)|^2 |M(z)|^2 |W(z)|^2 \frac{1}{\phi_u(\omega)} \tag{2.7}$$

このようにプレフィルタ $L(z)$ を決定することで，モデルベース制御器設計法で設計した場合と VRFT で設計した場合で漸近的には等価な制御器を設計することができる。

しかし，VRFT では 1 組の入出力データ $\{u_0(t),\ y_0(t)\}$ から 1 つの制御器を設計するため，データの特性に合わせて複数の区分的線形制御器を設計するためには，入出力データをそれぞれのモードごとに用意する必要がある。そこで，取得したデータを動特性のごとに自動的に分類するためにクラスタリングを利用する。

2.2　クラスタリング

クラスタリングとは特徴空間上での「距離」が近いデータを類似性の高いデータとして分類することで，元のデータセットを部分集合（クラスタ）に分ける手法のことである。クラスタリングは教師なし学習であるため，未知のデータに対しても類似したデータのみを集めることができる。

クラスタリングの手法には大きく分けて分割最適化手法と階層的手法の 2 つが存在する[23]。k-means 法は分割最適化手法の代表例である。k-means 法は $\mathcal{O}(Nk)$ であるため計算コストに優れるが，初期値依存性やクラスタ数の決定が試行錯誤的になるなどの問題がある。初期値依存性については，k-means++ 法[25] などでは改善されているものの，クラスタ数の決定は依然難しい。一方で，階層的手法ではクラスタ間距離関数に基づいてクラスタを一旦全て併合する。そして，この併合結果を階層構造（デンドログラム）にまとめて表示し，その上で与えたクラスタ数に実際に分類する。そのため，計算コストは $\mathcal{O}(N^2)$ に増加するものの，初期値依存性はない。さらに，クラスタの分割数をデンドログラムの情報を確認したうえで，設計者が後から決定可能である。よって，本研究ではデンドログラムからクラスタ数を決定できる階層的クラスタリングを採用することとし，これについて本節で説明する。

2.2.1　階層型クラスタリング

特徴ベクトル $v_i(i = 1, \ldots, N_\mathrm{S})$ を重心 $\boldsymbol{c}_j$ を持つ複数のクラスタ $\mathcal{C}_j(j = 1, \ldots, N_C)$ に分類する問題を考える。階層型クラスタリングではクラスタ間距離関数 $D(\mathcal{C}_1, \mathcal{C}_2)$ に基づいて，データを凝縮することで分類を行う。このクラスタ間距離関数 $D(\mathcal{C}_1, \mathcal{C}_2)$ の定義は様々提案されており，その選択によりクラスタリング結果は変化する。代表的な距離関数の定義を以下に示す。

単連結法:

$$D(\mathcal{C}_1, \mathcal{C}_2) = \min_{\boldsymbol{v}_1 \in \mathcal{C}_1, \boldsymbol{v}_2 \in \mathcal{C}_2} d(\boldsymbol{v}_1, \boldsymbol{v}_2)$$

完全連結法:

$$D(\mathcal{C}_1, \mathcal{C}_2) = \max_{\boldsymbol{v}_1 \in \mathcal{C}_1, \boldsymbol{v}_2 \in \mathcal{C}_2} d(\boldsymbol{v}_1, \boldsymbol{v}_2)$$

群平均法:

$$D(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{n_1 n_2} \sum_{\boldsymbol{v}_1 \in \mathcal{C}_1} \sum_{\boldsymbol{v}_2 \in \mathcal{C}_2} d(\boldsymbol{v}_1, \boldsymbol{v}_1)$$

Ward 法:

$$D(\mathcal{C}_1, \mathcal{C}_2) = E(\mathcal{C}_1 \cup \mathcal{C}_2) - E(\mathcal{C}_1) - E(\mathcal{C}_2)$$

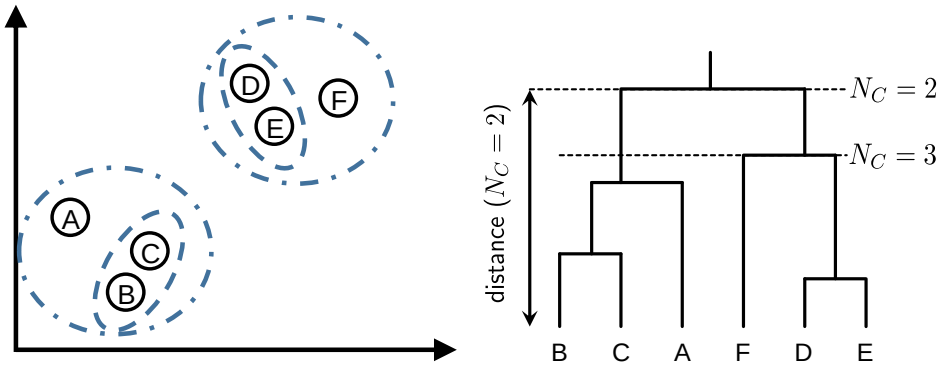
ここで，$d(\boldsymbol{v}_1, \boldsymbol{v}_2)$ はベクトル間距離を表す。また，$E(\mathcal{C}_j) = \sum_{\boldsymbol{v} \in \mathcal{C}_j} \left(d(\boldsymbol{v}, \boldsymbol{c}_j)\right)^2$ を満たす。ベクトル間距離関数 $d(\boldsymbol{v}_1, \boldsymbol{v}_2)$ にもマンハッタン距離などいくつかのバリエーションがあるが，典型的にはユークリッド距離が用いられる。このようにしてクラスタ間距離を定義することで，データ群の類似度を距離に基づいて評価することができる。

次に，データの凝縮について説明する。階層的手法では，初めに全てのデータ点をクラスタとみなし，クラスタが 1 つになるまでクラスタ間距離の近いクラスタを一旦併合する。図 2.4 に示すように，クラスタを凝縮していく過程を図示したものをデンドログラムと呼ぶ。図 2.4(b) において，横棒の高さは下にぶら下がっている 2 つのクラスタのクラスタ間距離 D を表す。例えば，D–E の距離が最も近く，次に近いのが B–C の距離，そして 3 番目に近いのが BC が含まれるクラスタ BC–クラスタ A の距離である。また，一番上の横棒の地点ではクラスタの総数 $N_C = 2$ であることが分かる。

デンドログラムを読み取ることで，不自然に遠いクラスタを併合しないようなクラスタ数を設定できる。従って，最適な制御器モードの数が未知であっても，デンドログラムの情報から適切なクラスタ数を決定することが可能となる。

2.3 Support Vector Machine

本節では，オンラインデータから接触環境の特性を識別するために用いる Support Vector Machine (SVM)[24] について説明する。SVM とは 2 つのクラスのデータ間のマージンを最大にするような識別関数を求める手法である。なお，学習に用いるデータは事前に分類されている必要があるため，SVM は教師あり学習の一種である。しかし，先にクラスタリングを用いてデータを分類しておけば，本研究にも適用可能である。



(a) クラスタの分布　　(b) デンドログラム

図 2.4: 階層的手法におけるクラスタの凝縮

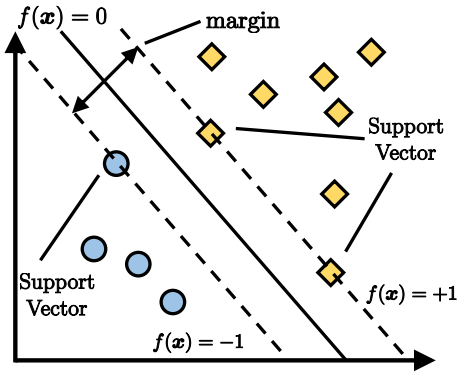


図 2.5: データと識別関数

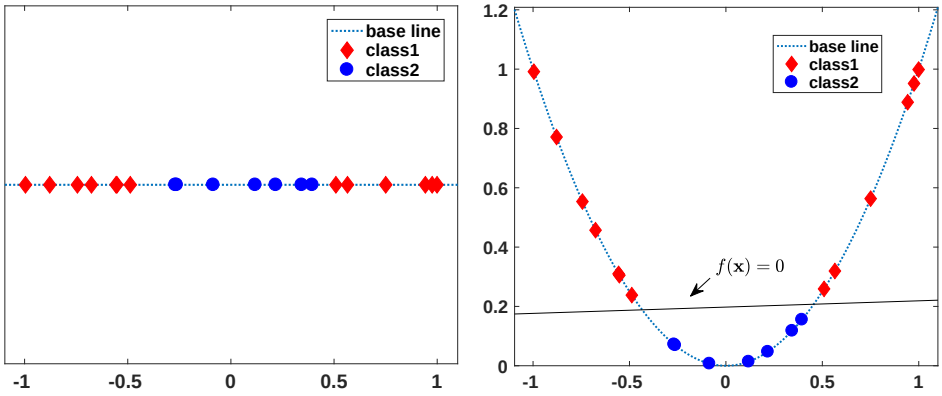
2.3.1　識別関数とマージン最小化

ラベル $l_i \in \{-1, 1\}$ をもつ教師付き学習データ $\boldsymbol{x}_i (i = 1, \ldots, N_m)$ が与えられるとする。この時，図 2.5 のような 2 クラス分類を考える。今，特徴空間 $\boldsymbol{x} \in R^n$ における識別関数 $f(\boldsymbol{x})$ を (2.20) 式で定義する。

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) + b \tag{2.20}$$

ここで，$\boldsymbol{w}$, b は学習パラメータ，$\boldsymbol{\phi}(\boldsymbol{x})$ は $\boldsymbol{x}$ を高次空間に写像する写像関数である。写像関数は非線形な識別問題を解く場合に重要となるが，詳しくは 2.3.2 説にて後述する。

識別関数を定義すると，図 2.5 のように，$f(\boldsymbol{x}) = 0$ がクラス間の境界を表す。また，クラス境界 $f(\boldsymbol{x}) = 0$ に最も近い特徴ベクトルをサポートベクタと呼び，$f(\boldsymbol{x}) = \pm 1$ 上に定義する。この時，クラス間のマージン μ は $f(\boldsymbol{x}) = \pm 1$ 間の距離で定義されるため，



(a) 元の空間 ($\boldsymbol{x} = [x_1]^{\mathrm{T}}$)　　(b) 写像空間 ($\boldsymbol{\phi}(\boldsymbol{x}) = [x_1, {x_1}^2]^{\mathrm{T}}$)

図 2.6: 写像関数を用いた非線形識別

(2.21) 式のように表せる。

$$\mu = \frac{2}{||\boldsymbol{w}||} \tag{2.21}$$

ここで，マージンが大きいことは汎化性能の観点でメリットがあるため，マージンを最大化する問題を考える。これにより，一般には識別関数の取り方は無限に存在するが，一意な識別関数を決定できる。よって，以下の最適化問題を解くことで識別関数を導出する。

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{b}} \quad & \frac{||\boldsymbol{w}||^2}{2} + \lambda \sum_{\forall i} \eta_i \\ \text{subject to} \quad & l_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + \boldsymbol{b}) \geq 1 - \eta_i \quad \forall i \end{aligned} \tag{2.22}$$

ここで，$\lambda > 0$ は設計者が与えるペナルティ項の重み，$\eta_i > 0$ はスラック変数である。ペナルティ項の重み λ は分類誤りに対するペナルティの大きさを指定するために用いる。この重み λ を適切に設定することで，マージン内へのデータの存在を許容させ，ノイズなどを含むデータに対してもロバストな識別器を得られる。

2.3.2　カーネルトリックによる非線形識別

本節では (2.20) 式で用いられる写像関数 $\boldsymbol{\phi}(\boldsymbol{x})$ とカーネル関数について説明する。

写像関数 $\boldsymbol{\phi}(\boldsymbol{x})$ の選択は非線形識別問題の場合には重要となる。例えば，図 2.5 のように，超平面（直線）でデータを分割できる場合には，線形識別問題である。線形識別問題では，$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$ としておけばよい。しかし，図 2.6(a) のような場合には，超平面（1 点）で分離できないため非線形識別問題となる。ここで，$\boldsymbol{\phi}(\boldsymbol{x}) = [x_1, {x_1}^2]^{\mathrm{T}}$ という写像関数

を取ると，図 2.6(b) のような空間に写像される。この写像空間上ではデータは線形分離できることが分かる。このように，非線形識別問題であっても写像空間上では線形識別可能であることが分かる。

実際に SVM を利用する際には，写像関数そのものではなくその内積であるカーネル関数 $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\phi}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})$ を利用して最適化計算を行う。以下のようなカーネル関数は広く用いられている[26]。

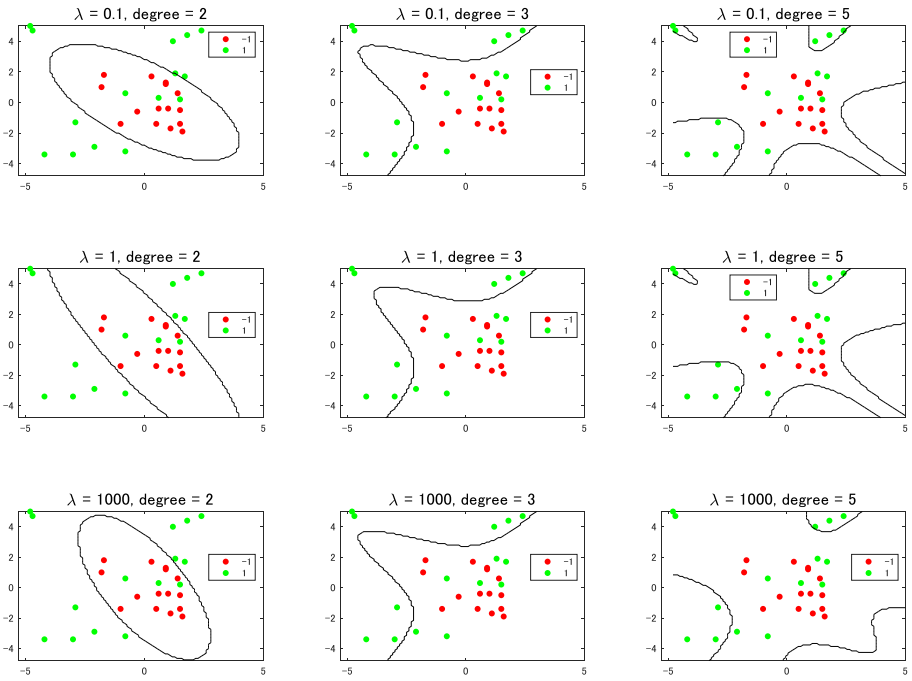
線形カーネル:	$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = {\boldsymbol{x}_1}^{\mathrm{T}}\ \boldsymbol{x}_2$
多項式カーネル:	$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\gamma + {\boldsymbol{x}_1}^{\mathrm{T}}\ \boldsymbol{x}_2)^d$
RBF カーネル:	$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\gamma \Vert \boldsymbol{x}_1 - \boldsymbol{x}_2 \Vert_2^2\right)$
シグモイドカーネル:	$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(\gamma {\boldsymbol{x}_1}^{\mathrm{T}}\ \boldsymbol{x}_2 + r)$

ここで，γ, d, r は設計者が与えるカーネルパラメータである。文献 [26] によれば，特に理由がなければ一般に RBF カーネルが最も妥当な選択であると述べられている。なぜなら，RBF カーネルが他のカーネルの一般形とみなせ，設計者が与えるべきハイパーパラメータが少なく，有限な数値で計算可能であるからである。それに対して，特徴ベクトルの数やその次元数が多く計算コストが莫大になる場合のみ線形カーネルの利用を検討するように提案されている。

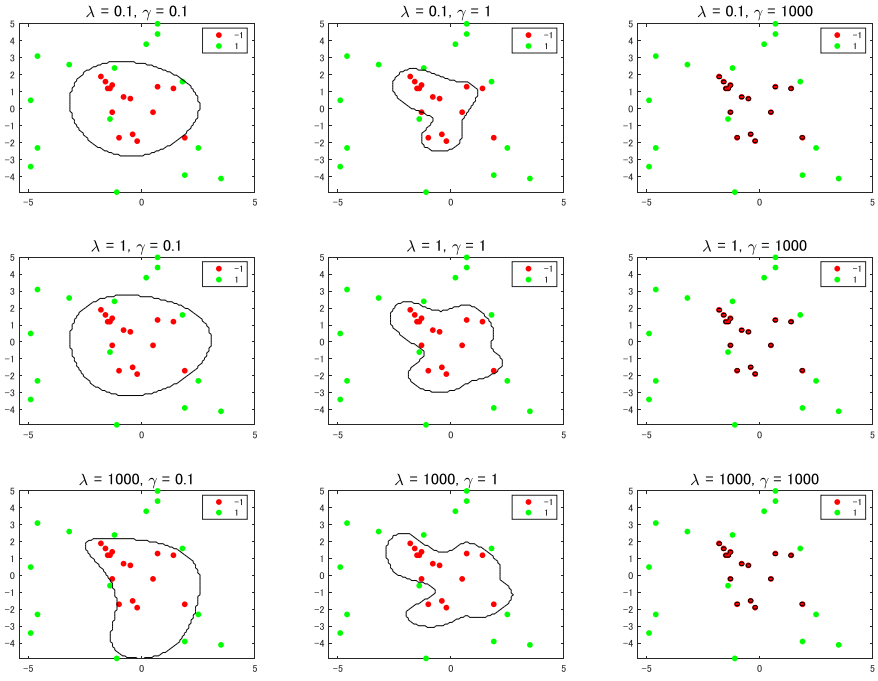
実際にランダムデータに対して多項式カーネルと RBF カーネルを適用した学習例を図 2.7 に示す。図 2.7(a) と図 2.7(b) を比較すると，複雑さを表すパラメータ degree や γ を大きくすると RBF カーネルの方がより自然な境界を学習できていることが分かる。その一方で，RBF カーネルで $\gamma = 1000$ の場合には過学習を起こしてしまっている。そのため，これらの設計者が与えるパラメータの設定には注意が必要である。

2.3.3　SVM パラメータの探索

コストパラメータ λ やカーネルパラメータ γ, d などの設計者が与えるべきパラメータをハイパーパラメータと呼ぶ。ハイパーパラメータの選択には，ある程度のヒューリスティックスを必要とする。ハイパーパラメータの探索手法としてはグリッドサーチやベイズ最適化に基づく手法[27] などが存在する。これらの詳細については本論文の範囲を超えるため，各文献を参照されたい。



(a) 多項式カーネルを用いた場合



(b) RBF カーネルを用いた場合

図 2.7: カーネル関数の選択とパラメータの影響

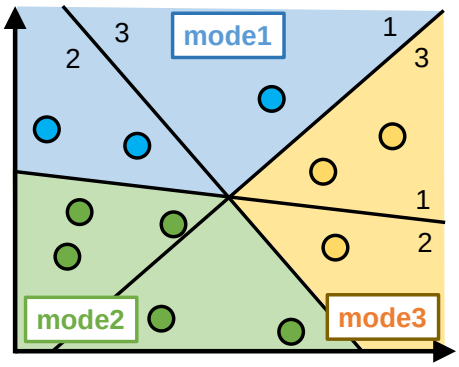


図 2.8: one-against-one によるマルチクラス SVM

2.3.4　マルチクラス SVM

最後に，3 つ以上のクラスを識別するためのマルチクラス SVM について説明する。ここまでは 2 クラス識別問題に限って説明してきたが，これを複数回適用することでマルチクラス識別を行うこともできる。

マルチクラス SVM の実装には，one-against-one(一対一) と one-against-the-rest（一対他）がある。図 2.8 のように，one-against-one アプローチでは，N_C 個のクラスに対して $\frac{N_C(N_C-1)}{2}$ 個の 2 クラス識別器を用意し，多数決で識別を行う。oen-against-the-rest アプローチでも，N_C 個の 2 クラス識別器を用意し多数決を行う。

本研究では，広く利用されている SVM のライブラリである libsvm[24] でも採用されている one-against-one アプローチを用いてマルチクラス分類を行う。これにより，環境が多数存在する場合においても，適応的制御器の設計が可能である。

第3章　切替型制御器の設計法

本章では，文献 [15] に基づいて，適応的制御器の一種である切替型制御器の設計法について述べる。まず，切替型制御器の構造と数学的表現について述べ，切替型制御器の状態量補償とチャタリング防止について説明する。そして，2 章で述べた VRFT とクラスタリング，SVM をどのように利用して設計するかを説明する。その後，従来手法をより一般的化するにカーネルを変更する必要があることを述べる。

3.1　切替型制御器の数学的表現

本節では，切替型制御器を数学的にどのように表現するかを説明する。切替型制御器の構造を図 3.1 に示す。このように切替型制御器は，複数の区分的線形制御器（候補制御器）とオンラインデータから切替条件を判別する部分 (スーパバイザ) から構成できる。ここで候補制御器の数（モード数）が n 個の切替型制御器は (3.1) 式のような PWA システムとして記述できる。

$$u(t) = \begin{cases} {\boldsymbol{\rho}_1}^{\mathrm{T}}\ \boldsymbol{\varphi}(t) & (\text{mode } 1) \\ {\boldsymbol{\rho}_2}^{\mathrm{T}}\ \boldsymbol{\varphi}(t) & (\text{mode } 2) \\ \quad\vdots & \quad\vdots \\ {\boldsymbol{\rho}_n}^{\mathrm{T}}\ \boldsymbol{\varphi}(t) & (\text{mode } n) \end{cases} \tag{3.1}$$

ここで $\boldsymbol{\rho}_\bullet$ は制御器パラメータベクトル，$\boldsymbol{\varphi}(t)$ は $\boldsymbol{\varphi}(t) = \boldsymbol{\beta}(z)\ e(t)$ として生成される制御器内の信号である。

切替型制御器全体は非線形なシステムであるが，(3.1) 式の各モードの候補制御器は線形時不変な制御器である。さらに，候補制御器はパラメータに対して線形に表現できる。そのため，クラスタリングを用いてモードごとに分類したデータを用意することで，モードごとに VRFT を適用することで設計が可能である。

切替型制御器では，制御対象の非線形特性に合わせて制御則を切り替える。これにより，非線形な制御対象に対しての性能や安定性の向上が期待できる。しかし，切替時には制御器の出力が不連続となるため，制御系に悪影響を与える可能性がある。そこで，制御器の状態量を補償することで，切替時にも制御器の出力が連続となるようにする。

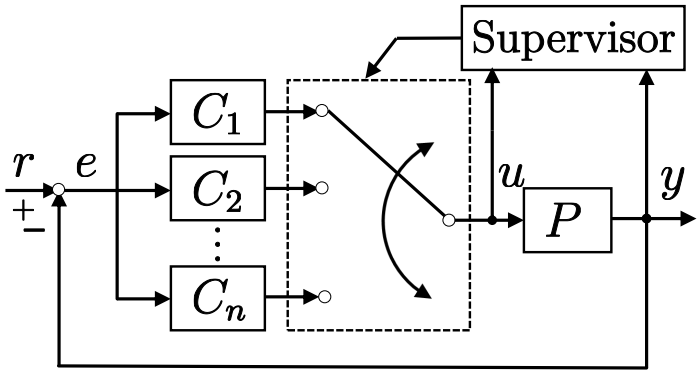


図 3.1: 切替型制御器

3.2　切替時の制御器状態量補償

本節では，モード切替時の状態量補償について説明する。切替時の状態量補償には，制御対象の状態量に注目する手法[22] やバンプレス制御[21] などがある。しかし，本設計法はデータ駆動型制御に基づく設計法であるため，制御対象の状態量を用いることは困難である。そこで，モード切替時に制御器の出力を連続にするようなバンプレス状態量補償を行う。

切替型制御器が入力 $u_{\mathrm{sw}}(t_{\mathrm{sw}})$，出力 $y_{\mathrm{sw}}(t_{\mathrm{sw}})$ である時刻 t_{sw} において，別の候補制御器制御器 $C_{\mathrm{next}}(z)$ へのモード切替が起こるとする。ここで，切替後の候補制御器 C_{next} の状態方程式を (3.2) 式とする。

$$\begin{cases} \boldsymbol{x}(k+1) = A\boldsymbol{x}(t) + Bu(t) \\ \quad y(t) = C\boldsymbol{x}(t) + Du(t) \end{cases} \tag{3.2}$$

この時, 切替の前後で制御器の出力が一致する状態量 $\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}})$ を考える。まず, $y_{\mathrm{sw}}(t_{\mathrm{sw}}) = y(t_{\mathrm{sw}})$ となる状態量は (3.3) 式を満たす。

$$C\boldsymbol{x}(t_{\mathrm{sw}}) = y_{\mathrm{sw}}(t_{\mathrm{sw}}) - Du_{\mathrm{sw}}(t_{\mathrm{sw}}) \tag{3.3}$$

状態量が 1 つであれば，(3.3) 式から状態量を算出できる。

状態量が 2 つの時には，さらに $y_{\mathrm{sw}}(t_{\mathrm{sw}}-1) = y(t_{\mathrm{sw}}-1)$ についても満たすことができる。この条件に (3.2) 式を代入し整理することで，(3.4) 式を得られる。

$$CA^{-1}\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}}) = y_{\mathrm{sw}}(t-1) - (D - CA^{-1}B)u_{\mathrm{sw}}(t_{\mathrm{sw}}) \tag{3.4}$$

よって，状態量が 2 つの場合には (3.3) 式と (3.4) 式を同時に満たす $\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}})$ を求めればよい。

上記の議論を一般化すると，状態量が n_x 個ある場合に n_x サンプル前までの制御器出力を一致させるためには，以下の式を満たす $\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}})$ を求めればよい。

$$U\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}}) = \boldsymbol{y}_{\mathrm{sw}}(t_{\mathrm{sw}}) - V\boldsymbol{u}_{\mathrm{sw}}(t_{\mathrm{sw}}) \tag{3.5}$$

ここで，行列 U, V とベクトル $\boldsymbol{u}_{\mathrm{sw}}(t_{\mathrm{sw}})$, $\boldsymbol{y}_{\mathrm{sw}}(t_{\mathrm{sw}})$ は以下のように定義される。

$$U = \begin{bmatrix} C & CA^{-1} & CA^{-2} & \dots & CA^{1-n_x} \end{bmatrix}^{\mathrm{T}} \tag{3.6}$$

$$V = \begin{bmatrix} D & 0 & 0 & \dots & 0 \\ 0 & D - CA^{-1}B & 0 & \dots & \vdots \\ 0 & D - CA^{-2}B & D - CA^{-1}B & \dots & \vdots \\ 0 & \vdots & \vdots & \ddots & 0 \\ 0 & D - CA^{1-n_x}B & D - CA^{2-n_x}B & \dots & D - CA^{-1}B \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{y}_{\mathrm{sw}} = \begin{bmatrix} y_{\mathrm{sw}}(t_{\mathrm{sw}}) & y_{\mathrm{sw}}(t_{\mathrm{sw}} - 1) & \dots & y_{\mathrm{sw}}(t_{\mathrm{sw}} - (n_x + 1)) & \end{bmatrix}^{\mathrm{T}} \tag{3.8}$$

$$\boldsymbol{u}_{\mathrm{sw}} = \begin{bmatrix} u_{\mathrm{sw}}(t_{\mathrm{sw}}) & u_{\mathrm{sw}}(t_{\mathrm{sw}} - 1) & \dots & u_{\mathrm{sw}}(t_{\mathrm{sw}} - (n_x + 1)) & \end{bmatrix}^{\mathrm{T}} \tag{3.9}$$

さらに，行列 U は正方行列であるため，この行列が正則であるとき (3.10) 式で状態量 $\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}})$ を決定できる。

$$\boldsymbol{x}_{\mathrm{sw}}(t_{\mathrm{sw}}) = U^{-1}\ \boldsymbol{y}(t_{\mathrm{sw}}) - U^{-1}V\boldsymbol{u}_{\mathrm{sw}}(t_{\mathrm{sw}}) \tag{3.10}$$

モード切替時には，(3.10) 式を用いて算出した状態量を次の候補制御器の状態量として再設定する。これにより切替時にも制御器の出力の急激な変化を防ぐことができる。

しかし，n_x サンプルの間に切替が複数回起こる場合には，複数のモードの出力をフィッティングしてしまい不自然な応答となる場合がある。そこで，前回の切替から $n_{\mathrm{sw}} < n_x$ サンプルの時点で切り替わった場合には，疑似逆行列を用いて n_{sw} サンプル前までの応答のみを一致させることとする。これにより，切替タイミングによらず制御器出力を滑らかに遷移させることができる。

3.3　切替時のチャタリングの防止

本節では，切替時のチャタリングの防止について説明する。状態量の補償によって低減は可能ではあるが，切替の影響で切替が発生してしまう可能性がある。このようなチャタリングを防止するために，切替条件に SVM のマージンを用いてヒステリシス特性を持たせる。これにより，制御器の切替を最小限に抑え，チャタリングを防止することができる。

3.4　切替型制御器の設計方法

本節では，切替型制御器の設計法の詳細について説明する。設計方法は時系列データをモードごとに分類し，候補制御器を設計する **Step1**〜**Step6** と，オンラインでの切替動作に関わる **Step7**〜**Step8** で構成される。

Step1: データ取得

制御対象から N サンプルの入出力データ $\{u_0(t), y_0(t)\}$ を取得し，(2.11) 式と (2.8) 式を用いてデータセット $\mathcal{D} = \{\boldsymbol{\varphi}(t), u_L(t)\}$ を算出する。

Step2: データ空間での分割

データセットを N_S 個のサブセット $\mathcal{S}_j$ $(j = 1, \ldots, N_\mathrm{S})$ にランダムに振り分ける。この時，$\mathcal{S}_j$ はデータセット内のランダムなデータからユークリッド距離の近い物を n_s 個集めたものである。ただし，$n_\mathrm{s} = N/N_S$ の関係である。
ここで，(2.17) 式の最適化問題を特徴づけるデータ空間で距離が近いデータは，近い制御器モードに属するデータと考えられる。よって，この作業は近い制御器を作るであろうデータ組を大量に作ることを意味する。

Step3: パラメータベクトル推定

分類されたそれぞれのサブセット $\mathcal{S}_j$ に対してパラメータベクトル $\hat{\boldsymbol{\rho}}_j$ を推定する。最小二乗法よりパラメータベクトル $\hat{\boldsymbol{\rho}}_j$ は (3.11) 式で推定できる。

$$\hat{\boldsymbol{\rho}}_j = \left({\Phi_j}^\mathrm{T}\Phi_j\right)^{-1}\Phi_j\boldsymbol{U}_j \tag{3.11}$$

ただし，$\Phi_j = [\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_{n_\mathrm{s}}]^\mathrm{T}$, $\boldsymbol{U}_j = [u_1, \ldots, u_{n_\mathrm{s}}]^\mathrm{T}$ である。
ここで，(3.11) 式は VRFT の最適化問題 (2.17) に対応する。つまり，この作業は VRFT によって，区分的な制御器を大量に設計することを意味する。

Step4: パラメータ空間でのクラスタリング

Step3 で求めた $\hat{\boldsymbol{\rho}}_j$ をクラスタリングで N_C 個のクラスタに分類する。ただし，クラスタ数 N_C はデンドログラムの結果より設計者が決定する。
パラメータ空間は制御器のゲインを表すため，この空間における類似度は近い区分的線形制御器を表す。つまり，この作業は大量に作成した区分的線形制御器を支配的ないくつかのモードにまとめる事を意味する。

Step5: データ空間へのクラスタリング結果の反映

パラメータ空間でのクラスタリング結果に基づいて，元のデータセット $\mathcal{D}$ を各モードのデータセット $\mathcal{D}_m$ $(m = 1, \ldots, N_C)$ に振り分ける。
これにより，VRFT の設計に用いるデータセットそのものを分類することができる。

Step6: 候補制御器の設計

分類されたデータセット $\mathcal{D}_m$ に対して VRFT を用いてモードごとの候補制御器 $C_m(z)$ を設計する。

Step7: 切替条件の決定

分類されたデータセット $\mathcal{D}_m$ に対して SVM を用いてデータ空間での識別器を学習させる。
データ空間上での位置はオンラインにおいても毎サンプル計算可能である。したがって，推定を伴う方法と異なり，遅延なく制御器モードの判定が行える。

Step8: 初期状態量補償付き切替型制御器

Step6 で設計した候補制御器と **Step7** で導出した切替条件を実装し，バンプレス状態量補償付きの切替型制御器を構成する。

以上のようにオフラインで設計しておくことで，オンラインでは **Step7** で導出した識別器に基づいて，オンラインデータから **Step6** で設計した候補制御器の切り替えを行うことができる。これにより，常に現在の制御対象に適した候補制御器を用いることとなり，制御対象の非線形性へ対応することができる。さらに，候補制御器の設計は VRFT，設計される制御器の動特性ごとのデータ分類はクラスタリング，識別器は SVM をそれぞれ用いることで，システマティックな設計法とすることができる。そのため，設計者の負担を最小限に抑えることができる。

3.5　一般的な制御対象への拡張

本節では，文献[15] の手法を一般化する際の注意点について述べる。文献 [15] では，出力 $y(t)$ によって連続的にパラメータが変化する非線形システムである 1 リンク鉛直型アームを制御対象としていた。また，同じ著者による文献 [28] では，$y(t)$ によって離散的にモードが切り替わるシステムであるハードディスクのヘッドを制御対象とした設計例は存在する。しかし，一般には $y(t)$ のみによってモードできるとは限らない。

そこで，本研究では切替条件が複雑でも対応できるように，**Step7** における SVM のカーネル関数を RBF カーネルへ変更することを提案する。先行研究では，カーネル関数として多項式カーネルを用いていた。しかし，図 2.7 で示したように，多項式カーネルでは複雑な境界を識別することは難しい。そのため，本研究で対象とする $u(t)$ と $y(t)$ の組み合わせで変化する一般的なシステムに対しては十分な識別を行うことは難しい。一方で，RBF カーネルでは過学習に注意を払う必要はあるものの，複雑なデータ分布に対しても識別も可能である。以上より，RBF カーネルを採用することで，複数の状態量で非線形性が定まるような一般のシステムにも設計法を拡張することができると考えられる。

第4章　ゲインスケジュールド制御器の設計

本章では，切替型制御器の設計法をゲインスケジュールド制御器の設計法へ拡張する方法について説明する。初めに，切替型制御の問題点について明らかにし，その解決策としてゲインスケジュールド制御器を導入する。そして，提案するオンラインでのスケジューリング則について述べる。

4.1　ゲインスケジュールド制御系の数学的表現

本節では，ゲインスケジュールド制御系の数学モデルについて説明する。

4.1.1　切替型制御系の問題点

3 章で述べた切替型制御器では，切替時にバンプレス状態量補償を行っていた。しかし，文献 [22] のように性能を評価した手法でないため，バンプレス状態補償では過渡応答の劣化の可能性がある。また，制御器の次数が増えると状態量の計算コストも増大してしまう。そこで，切替自体を行わずに制御対象の変化に適応するために，制御器のパラメータを連続的に変化させるゲインスケジュールド制御器を導入する。

4.2　ゲインスケジュールド制御器

提案するゲインスケジュールド制御器の構造を図 4.1 に示す。制御器はオフラインで設計した候補制御器 $C_n(z)$ とスーパーバイザによって調整される可変ゲイン $\theta_n(t) \geq 0$ で成り立つ。ここで，それぞれの候補制御器が (2.2) 式のように線形にパラメトライズされているならば，ゲインスケジュールド制御器は以下のように定式化できる。

$$u(t) = \boldsymbol{\varphi}^{\mathrm{T}}(t) \sum_{n=1}^{N_C} \theta_n(t) \boldsymbol{\rho}_n \tag{4.1}$$

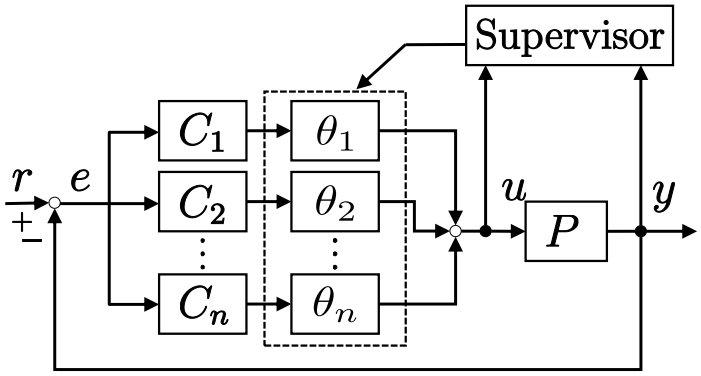


図 4.1: 提案するゲインスケジュールド制御器

ここで，$\boldsymbol{\varphi}(t)$ は $\boldsymbol{\varphi}(t) = \boldsymbol{\beta}(z)\ e(t)$ として生成される制御器内の信号である。また，可変ゲインは $\sum_{n=1}^{N_C} \theta_n = 1$ で正規化されているとする。

切替制御では制御器は瞬時に切り替わることしか出来なかったが，ゲインスケジュールド制御器では全体としての制御器出力 $u(t)$ は可変ゲインによって滑らかに変化することができる。そのため，切替制御のように制御器の内部状態量の補償は必要なく，様々な環境に対して適応可能であると考えられる。さらに，候補制御器の設計方法には切替型制御器の設計法がそのまま適用可能である。そのため，データ駆動型制御器設計法と機械学習による設計者の負担の低減など従来手法の利点をそのまま受け継いでいる。

4.3　スケジューリング則

ここでは，上記の構造をとった場合の可変ゲイン θ_n の決定法について述べる。SVM において，時刻 t のデータベクトル $\boldsymbol{x}(t)$ に対するラベル $l(t)$ は，学習済みの識別関数 $f(\boldsymbol{x})$ によって以下の式で識別される。

$$l(t) = \mathrm{sgn}(f(\boldsymbol{x}(t))) \tag{4.2}$$

$$\boldsymbol{x}(t) = \left[\boldsymbol{\varphi}^{\mathrm{T}}(t), u_L(t)\right]^{\mathrm{T}} \tag{4.3}$$

したがって，SVM の識別するラベルは $f(\boldsymbol{x}(t)) = 0$ を境に瞬時に切り替わることが分かる。しかし，SVM の定義から $|f(\boldsymbol{x}(t))| \leq 1$ の場合には，そのデータベクトル $\boldsymbol{x}(t)$ はマージン内に存在することが分かる。図 2.5 に示したように，基本的にマージン内には学習データが存在しないため，マージン領域は 2 つのクラスが共有する領域となる。つまり，このような場合には 2 つのクラス間を遷移する不確かな状態にあると見なせる。

そこで，マージン内ではマージンへの侵入割合によってそれぞれの候補制御器の寄与

度を決定することとする。つまり，モード数 $N_C = 2$ である場合，以下のように可変ゲイン $\boldsymbol{\theta}(t) = [\theta_1(t),\ \theta_2(t)]^{\mathrm{T}}$ を決定する。

$$\theta_1(t) = \begin{cases} 0 & (f(\boldsymbol{x}(t)) \leq -1) \\ \frac{f(\boldsymbol{x}(t))+1}{2} & (\text{otherwise}) \\ 1 & (f(\boldsymbol{x}(t)) \geq 1) \end{cases} \tag{4.4}$$

$$\theta_2(t) = 1 - \theta_1 \tag{4.5}$$

マルチクラスの場合もそれぞれの識別関数ごとに計算し正規化することで拡張可能である。例えば，$N_C = 3$ の場合には，3 つの識別関数から場合分けを行う。まず，それぞれの識別関数に関して (4.4) 式を用いて，2 クラス間の寄与度を計算する。つまり，モード i とモード j の間の識別関数が $f_{ij}(\boldsymbol{x})$ とすると，(4.6) 式を計算する

$$\begin{aligned} \theta_i^j(t) &= \begin{cases} 0 & (f_{ij}(\boldsymbol{x}(t)) \leq -1) \\ \frac{f_{ij}(\boldsymbol{x}(t))+1}{2} & (\text{otherwise}) \\ 1 & (f_{ij}(\boldsymbol{x}(t)) \geq 1) \end{cases} \\ \theta_j^i(t) &= 1 - \theta_i^j \end{aligned} \tag{4.6}$$

そして，(4.7) 式のように，それらの相加平均をそのモードの可変ゲインとして設定する。

$$\theta_i(t) = \frac{1}{N_{\mathrm{svm}}} \sum_{j=1}^{N_{\mathrm{svm}}} \theta_i^j \tag{4.7}$$

ここで，$N_{\mathrm{svm}} = \frac{N_C(N_C-1)}{2}$ である。

第5章　力制御系のモデリング

本章では，本研究で扱う制御対象である力制御系のモデリングについて述べる。本研究ではリニアモータを用いた 1 自由度ロボットによって力制御系を構成する。そこで，初めにリニアモータの特性と接触環境との相互作用を考慮することで力制御系のモデリングを行う。さらに，外乱に対してロバストな制御系を構築するために外乱オブザーバを付加し，環境以外の特性のノミナル化を行う。その上で，複数の状態量から決まる非線形性について述べる。

5.1　制御対象のモデリング

本節では，力制御系を構成するリニアモータの運動方程式と環境モデルについて述べる。リニアモータは供給した電力 $i_a(t)$ に比例した推力を発生する。よって，発生する推力は (5.1) 式のように表される。

$$f_t(t) = K_t i_a(t) \tag{5.1}$$

ここで，$f_t(t)$ はモータ推力，K_t は推力定数，$i_a(t)$ はリニアモータに供給した電力を表す。したがって，リニアモータの運動方程式は (5.2) 式のように表される。

$$m\ddot{x} = K_t i_a - f_{\mathrm{dis}} \tag{5.2}$$

ここで，m はリニアモータの質量，$x(t)$ はリニアモータの位置，$f_{\mathrm{dis}}(t)$ は外乱力を表す。外乱力 f_{dis} は環境からの反力 f_{res} と，摩擦力などの内力 f_{int} に分けられる。よって，(5.3) 式のように表せる。

$$f_{\mathrm{dis}} = f_{\mathrm{res}} + f_{\mathrm{int}} \tag{5.3}$$

次に，接触環境との相互作用について考える。一般に，接触する環境はバネ-マス-ダンパ系で表すことができる。よって，環境の動特性を環境インピーダンス $Z_e(s)$ として，(5.4) 式で定義する。

$$Z_e(s) = m_e s^2 + D_e s + K_e \tag{5.4}$$

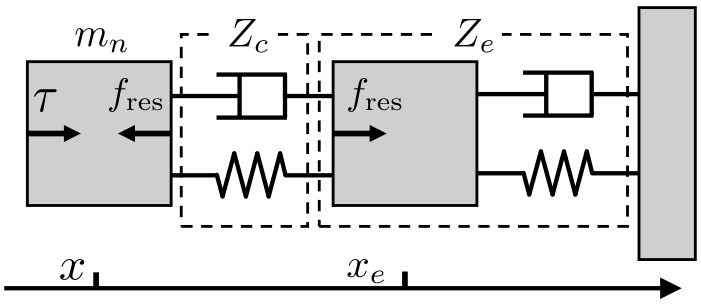


図 5.1: 接触環境との相互作用

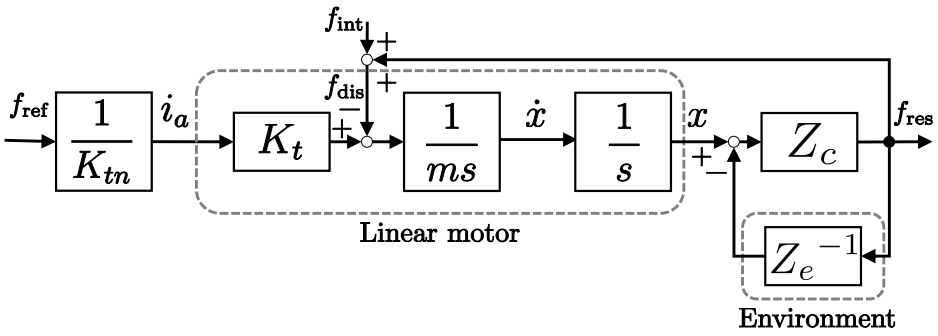


図 5.2: 制御対象

ここで，K_e は環境剛性，D_e は環境粘性，m_e は環境質量を表す。この時，ロボットと環境の相互関係は，図 5.1 となる。ただし，ロボットのエンドエフェクタ自体の機械インピーダンスを $Z_c(s)$，環境の表面位置を $x_e(t)$ とした。この時，環境反力 f_{res} は以下のように表せる。

$$f_{\mathrm{res}}(t) = Z_e(s)\ x_e(t) \tag{5.5}$$

$$f_{\mathrm{res}}(t) = Z_c(s)(x(t) - x_e(t)) \tag{5.6}$$

(5.5) 式と (5.6) 式より，$x_e(t)$ を消去すると，(5.7) 式を得る。

$$f_{\mathrm{res}}(t) = \frac{1}{{Z_c}^{-1}(s) + {Z_e}^{-1}(s)}\ x(t) \tag{5.7}$$

以上より，環境を含めた制御対象は図 5.2 のように構成される。ただし，下添え字 $\bullet_n$ はノミナル値を表す。

5.2　外乱オブザーバ

本節では，リニアモータ自体のロバスト性を高めるための外乱オブザーバ[29] について説明する。モーションコントロールでは，制御対象のパラメータ変動や摩擦などの不確かさを含む外乱によって制御性能が損なわれることがある。そこで，外乱に対してロバストな制御系を実現させる手法として外乱オブザーバが提案されている。外乱オブザーバとは，摩擦力，パラメータ変動，外力など制御対象に加わる外乱を外乱力として推定し，その推定した外乱力によって実際の外乱の影響を打ち消す制御アルゴリズムである。

(5.2) 式より，外乱力 f_{dis} は (5.8) のように表される。

$$f_{\mathrm{dis}}(t) = K_{tn} i_a(t) - m_n \ddot{x}(t) \tag{5.8}$$

よって，外乱力はエンコーダの位置情報を微分した加速度情報 $\ddot{x}$ と電流 i_a から推定することができる。ただし，位置情報や速度情報から加速度情報を得るためには微分動作が必要となるが，純粋微分はプロパーな演算ではない。よって，ローパスフィルタ $F(s)$ を用いた擬似微分を用いて計算を行うもののとする。したがって，推定外乱力 $\hat{f}_{\mathrm{dis}}$ は (5.9) 式で算出できる。

$$\begin{aligned} \hat{f}_{\mathrm{dis}} &= F(s)\ f_{\mathrm{dis}} \\ &= \frac{g_L}{s + g_L}\ (K_{tn} i_a - m_n \ddot{x}) \end{aligned} \tag{5.9}$$

ここで，g_L はローパスフィルタ $F(s)$ のカットオフ周波数である。

(5.9) 推定外乱力を打ち消すような電流を予め加えておくことで，外乱の影響を抑制できる。外乱オブザーバを加えた制御系を図 5.3 に示す。ここで，$\hat{\bullet}$ は推定値，下添え字 $\bullet_n$ はノミナル値，$i_{\mathrm{cmp}}(t)$ は補償電流を表す。外乱オブザーバを用いることで，出力を外乱によらず制御することが可能となる。

5.3　ノミナルプラントモデル

本節では，外乱オブザーバによって補償されたシステムのノミナルモデルを導出し，環境変動による非線形性を考える。外乱オブザーバが理想的に働くとき，図 5.3 のシステムは，図 5.4 のシステムと等価となる。ただし，f_{int} にはパラメータ変動分の等価外乱も含まれるとする。

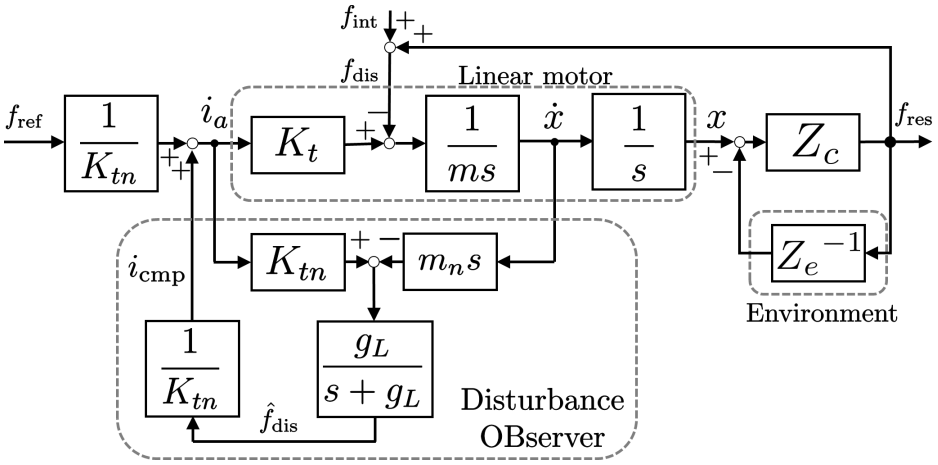


図 5.3: 外乱オブザーバによって補償された制御対象

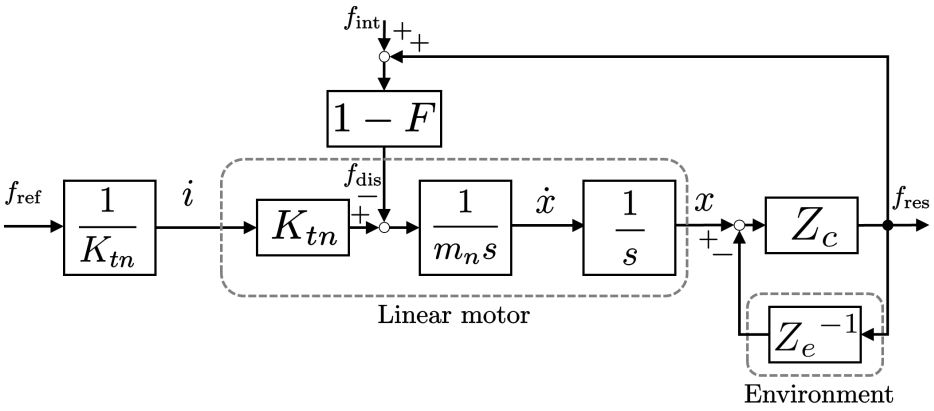


図 5.4: ノミナル制御対象

この時，制御対象の伝達関数は (5.10) 式となる。

$$P_n(s) = \frac{f_{\text{res}}(t)}{i(t)} = \frac{K_{tn}}{m_n s^2 (Z_c{}^{-1}(s) + Z_e{}^{-1}(s)) + (1 - F(s))} \tag{5.10}$$

(5.10) 式より，外乱オブザーバを用いたとしても Z_e の特性は補償されていないことが分かる。また，(5.7) 式より，環境 $Z_e(s)$ が変動した時，位置 x に対する反力 f_{res} が変動することが分かる。つまり，仮に同じ f_{res} が出力されていたとしても，環境が同一であるとは限らない。そのため，環境モードを判別するためには，位置 $x(t)$ の情報も同時に用いる必要がある。対して，文献 [15] での制御対象である鉛直アームはアーム角度 $\theta(t)$ のみで非線形特性が決まっていた。以上より，力制御系を対象とすることで，より一般的な複数の状態量を考慮しなければならない状況を検証できる。

第6章　シミュレーションによる検証

本章では，3 章と 4 章で述べた設計法を 5 章で述べた力制御系に対して適用する。これにより，適応的制御器が力制御系においても有効に働くことを検証する。

6.1　シミュレーション条件と設計条件

本節では，シミュレーション条件と設計条件を示す。なお，本章では簡単のため連続時間伝達関数として示すが，実際のシミュレーションでは双一次変換で離散化した伝達関数を用いた。また，設計した適応的制御器は Simulink を用いて実装した。

制御対象は 5 章でモデリングしたリニアモータによる力制御系とする。なお，反力応答値は真値にローパスフィルタを通した信号を用いた。シミュレーションパラメータは表 6.1 に示す。

2 つの環境に対して制御器を設計することとする。それぞれの環境インピーダンスは以下の通りに与えた。

$$\begin{cases} Z_{e1}(s) & = 0.75s^2 + 10\sqrt{15}s + 500 \\ Z_{e2}(s) & = 0.75s^2 + 100\sqrt{3}s + 10000 \end{cases} \tag{6.1}$$

次に設計に用いた時系列データについて述べる。入出力データはステップ信号（データ長 $N_1 = 200$）と M 系列信号（データ長 $N_2 = 81915$）を結合したものを用いた。ステップ信号は，エンドエフェクタと環境を接触させた状態でデータを取得させるために印加した。M 系列信号は，5 周期 14 段の M 系列信号を用いた。この信号をそれぞれの環境において印加し，入出力データを取得した。また，それぞれのデータについて，(2.11) 式と (2.8) 式を用いてデータセットを生成した。これら 2 つの環境分のデータを 1 周期ずつ切り出し，結合させたデータ（データ数 $N = 32766$）を設計用のデータセットに用いた。

次に候補制御器の設計条件について述べる。参照モデル $M(s)$ と周波数重み $W(s)$ は

以下で与えた。

$$M(s) = \left(\frac{100}{s+100}\right)^3 \tag{6.2}$$

$$W(s) = \frac{1}{s} \tag{6.3}$$

ここで，$M(s)$ の次数は (5.10) 式の制御対象の相対次数が 2 であることから決定した。

また，設計する候補制御器の構造は，(6.4) 式で与えた。

$$\boldsymbol{\beta}(s) = \begin{bmatrix}\beta_4(s) & \beta_3(s) & \beta_2(s) & \beta_1(s) & \beta_0(s)\end{bmatrix}^{\mathrm{T}} \; N(s)\; F_M(s) \tag{6.4}$$

$$\beta_4(s) = \frac{20s}{s+20}\cdot\beta_3(s), \qquad \beta_3(s) = \frac{40s}{s+40}\cdot\beta_2(s),$$
$$\beta_2(s) = \frac{60s}{s+60}\cdot\beta_1(s), \qquad \beta_1(s) = \frac{80s}{s+80}\cdot\beta_0(s), \qquad \beta_0(s) = 1$$

ここで，$F_M(s)$ は (2.15) 式の $\frac{M}{1-M}$ の分母を作るためのローパスフィルタであり，(6.5) 式で与えられる。

$$F_M(s) = \frac{30000}{s^2+300s+30000} \tag{6.5}$$

また，$N(s)$ は機械系の共振特性を打ち消すためのノッチフィルタであり，(6.6) 式で与えられる。

$$N(s) = \frac{s^2+\delta\cdot 2\zeta\omega_n s+{\omega_n}^2}{s^2+2\zeta\omega_n s+{\omega_n}^2} \tag{6.6}$$

今回は，$\delta = 0.5,\ \zeta = 5,\ \omega_n = 1200$ とした。また，双一次変換による離散化の際には周波数プリワーピング処理を施した[30]。

次にデータ分類について条件を述べる。サブセットの数は $N_S \quad = 762$（それぞれのサブセットの要素数は $n_s = 43$）とした。また，クラスタリングには Ward 法を用いた。なお，クラスタ数 N_C はデンドログラムの結果を見て決定することとする。

最後に，識別器についての条件を述べる。SVM のカーネル関数には RBF カーネルを用いた。また，ハイパーパラメータであるコストパラメータ η とカーネルパラメータ γ は，試行錯誤的に以下のようにした。

$$\eta = 1000 \tag{6.7}$$

$$\gamma = 0.01 \tag{6.8}$$

以上の条件の下で適応的制御器の設計を行った。

表 6.1: シミュレーションパラメータ

K_t	33 N/A	推力定数
K_{tn}	33 N/A	推力定数のノミナル値
m	0.75 kg	リニアモータの質量
m_n	0.75 kg	リニアモータのノミナル質量
Z_c	700 kN/m	リニアモータのノミナル質量
g_L	200 rad/s	外乱オブザーバのカットオフ周波数
g_v	400 rad/s	速度推定器のカットオフ周波数
g_f	100 Hz	力センサのカットオフ周波数
T_s	1.0 ms	サンプリングタイム

6.2　候補制御器の設計

本節では，切替型制御器とゲインスケジュールド制御器の両者で用いられる候補制御器の設計結果を示す。

6.2.1　データの分類

まず，**Step1** から **Step4** を実施し，作成したデンドログラムを図 6.1 に示す。デンドログラムより，クラスタ数を 3 つから 2 つに減らしたとき，一気に大きなクラスタ間距離を確保できることが分かる。クラスタ間距離が大きいとき，2 つのクラスタは有意に異なる特性を持つ制御器パラメータの集合であると考えられる。したがって，$N_C = 2$ と決定することができる。この時，**Step5** まで実施すると，時系列データは図 6.2 のように分類される。図 6.2 では，サンプリング点を環境とモードごとに表示している。この結果より，mode1 には Z_{e1} 由来のデータが多く分類され，mode2 には Z_{e2} 由来のデータが多いことが読み取れる。それぞれの分類のデータ点の数をまとめたものを表 6.2 に示す。最も望ましい結果は表の非対角要素が 0 個となることであり，実際に対角要素と比べると少なくなっていることが確認できる。この結果より，各モードには動特性ごとにデータが分類されていることが確認できた。

さらに定量的に評価するために，適合率と再現率を算出する。適合率 (Precision) と再現率 (Recall) の定義はそれぞれ以下で与えられる。

$$\text{Precision} = \frac{T_P}{T_P + F_P} \tag{6.9}$$

$$\text{Recall} = \frac{T_N}{F_P + F_P} \tag{6.10}$$

ここで，T_P, F_N, F_P, T_N はそれぞれ True Positive, False Negative, False Positive, True Negative であるデータの個数を表す。適合率が高いということはその mode のデータの純度が高いことを示すため，候補制御器の設計が有利になると考えられる。一方で，再現率が高いということは環境の情報が適切に切り分けできていることを示すため，識別器の学習が有利になると考えられる。ただし，適合率と再現率にはトレードオフの関係があるため，これらのバランスを考慮するための指標として (6.11) 式で定義される F 値 (F-measure) を算出する。

$$\text{F-measure} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \tag{6.11}$$

Z_{e1} 由来のデータが mode1 に分類されるのが適切 (True Positive) と仮定すると，適合率と再現率は表 6.3 のようになる。この結果より，総合的に適切な分類が成されていることが確認できた。さらに，再現率の差が大きいことから mode2 の制御器が選ばれる確率が高いことが示唆される。

6.2.2　VRFT による候補制御器設計

次に，**Step6** において分類済みのデータセットから候補制御器を設計した結果を図 6.3 に示す。ここで，$C_m(z)$ $(m = 1,\ 2)$ は設計されたそれぞれのモードの候補制御器，$C_\bullet{}^*(z)$ は環境 $Z_{e\bullet}(z)$ に対する理想制御器である。この結果より，理想制御器に近い特性の制御器が得られていることが分かる。設計されたパラメータ $\boldsymbol{\rho}_m$ の値と評価値 $J_{\mathrm{VR}}(\boldsymbol{\rho}_m)$ を表 6.4 にまとめた。また，表 6.4 には比較として分類が 100% 正確な場合の VRFT の結果も併記した。表 6.4 より，通常の VRFT と近いパラメータを得られたことが分かる。以上より，提案手法を用いて候補制御器の設計が適切に行われたことを確認できた。

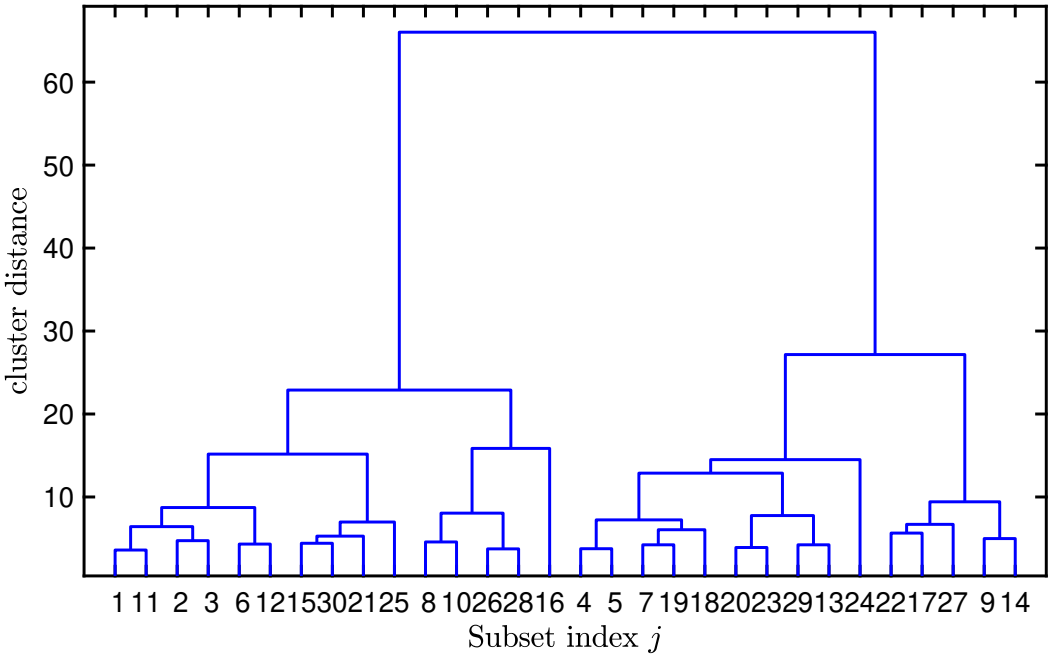


図 6.1: デンドログラム

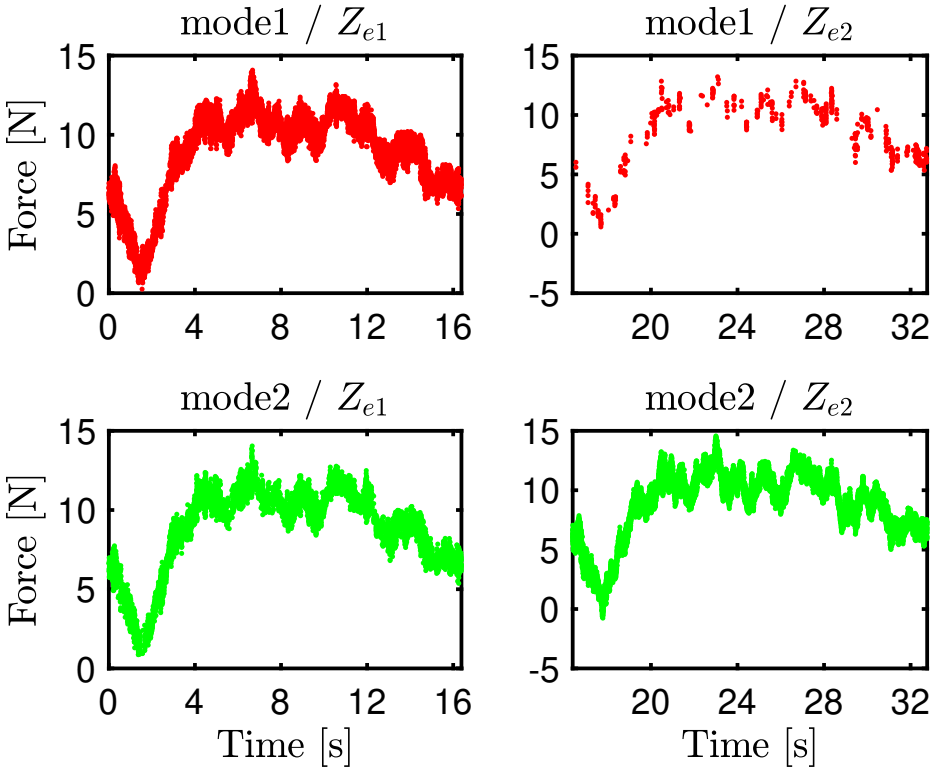


図 6.2: 時系列データの分類結果

表 6.2: 混同行列

	Z_{e1} 由来	Z_{e2} 由来	合計
mode1	10256	494	10750
mode2	6127	15889	22016
合計	16383	16383	32766

表 6.3: 分類の定量的評価

	適合率	再現率	F 値
mode1	95.4 %	62.6 %	75.6 %
mode2	72.2 %	97.0 %	82.8 %

表 6.4: 候補制御器の設計結果

データ元	mode1	mode2	Z_{e1}	Z_{e2}
ρ_1	-1.0563×10^{-7}	-2.2446×10^{-9}	-9.1391×10^{-8}	-5.6061×10^{-11}
ρ_2	3.7704×10^{-6}	7.2272×10^{-8}	3.4937×10^{-6}	1.9022×10^{-9}
ρ_3	-9.0386×10^{-5}	-1.8339×10^{-6}	-9.1048×10^{-5}	-2.6466×10^{-7}
ρ_4	1.3688×10^{-3}	8.0675×10^{-5}	1.4827×10^{-3}	6.0578×10^{-5}
ρ_5	6.9708×10^{-3}	6.6797×10^{-3}	6.6677×10^{-3}	6.6547×10^{-3}
J_{VR}^{N}	1.6238×10^{-10}	1.3236×10^{-10}	1.3805×10^{-11}	2.4948×10^{-12}

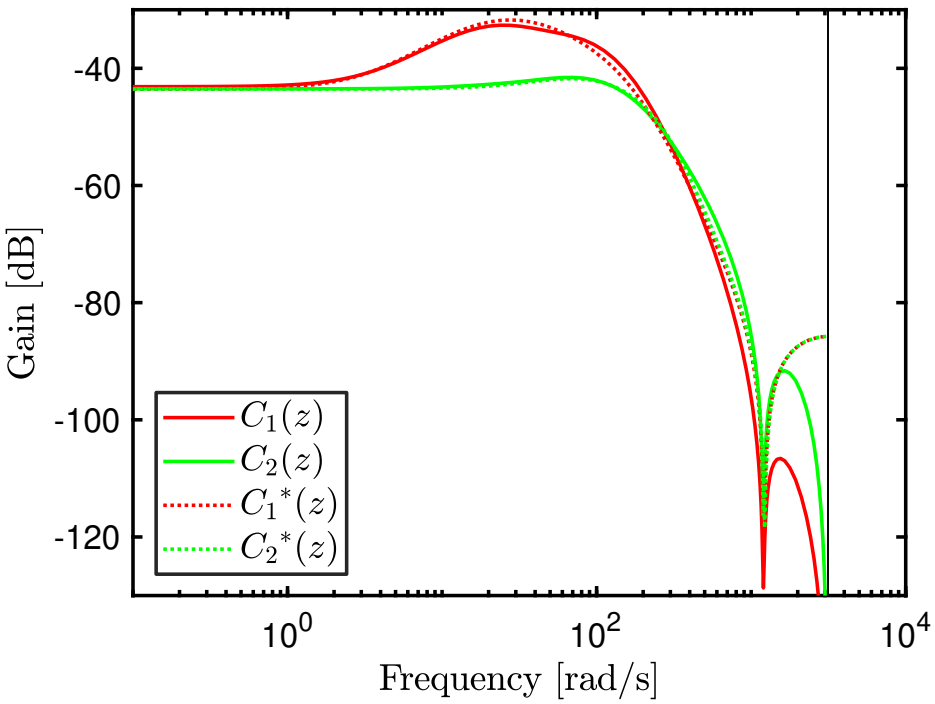


図 6.3: 候補制御器と理想制御器のボード線図

6.3　適応的制御器の時間応答

本節では，設計した候補制御器を用いて適応的制御器を実装したときの応答について検証を行う。ステップ応答は，時刻 1.0 s から 6.0 N の力指令値を印加した。**Step8** までを実施して得られた切替型制御器 $C_{\mathrm{SW}}(z)$ のステップ応答とモードの時間変化を図 6.4 に示す。また，第 4 章で提案したゲインスケジュールド制御器 $C_{\mathrm{GS}}(z)$ のステップ応答と可変ゲイン $\theta_1(t)$ の時間変化を図 6.5 に示す。シミュレーションはそれぞれの環境において行った。この時，どちらの環境においても安定かつ参照モデルに近い応答を実現する制御器が望ましい制御器であるといえる。しかし，固定の制御器ではどちらか片方の環境でしか所望の特性は実現できていないことが分かる。これに対して，適応的制御器が望ましい特性を実現可能であるかどうかを以下で検証していく。

図 6.4(a) では，時刻 1.17 s 前後から $Z_{e1}(z)$ に適した mode1 へ切替を行うことで適切な制御器を利用しようとしている。一方で，図 6.4(b) では，切替を行わないという判断をしていることが分かる。これにより，固定の制御器を利用した場合と比べて，Z_{e2} では振動を抑制できた。しかし，ステップ信号の印加から 0.17 s ほど切替までに遅れが発生してしまった。また，切替後の応答もゆっくりとしたもので，整定時間としては $C_2(z)$ を用いたものと大差ない結果となった。そのため，切替型制御器を用いたとしても Z_{e1} では性能の向上幅はわずかだった。

これに対して，図 6.5(a) では，1.17 s 前後まで mode2 の制御器が中心に使われており，切替型制御器と同様の挙動を示した。しかし，その後 1.23 s ごろまでの間では，2 つの候補制御器を補完するように可変ゲインを調整しており，結果として整定時間が短くなった。ただし，可変ゲインを急激に変化させたため，振幅は小さいものの高周波の振動が生じてしまった。また，図 6.5(b) では，切替型制御器と同様に振動を抑制できたことが確認できる。

次に，定量的に性能を評価するために，これらの応答と参照モデルの応答との平均 2 乗誤差 (Mean Square Error: MSE) を算出した。それぞれの制御器を用いた時の MSE を比較したものを表 6.5 に示す。ここで，それぞれの応答 $y(t)$ と参照モデルの応答 $y_d(t)$ の間の MSE は (6.12) 式で定義される。

$$\mathrm{MSE} = \frac{1}{N}\sum_{t=0}^{N}\left(y_d(t) - y(t)\right)^2 \tag{6.12}$$

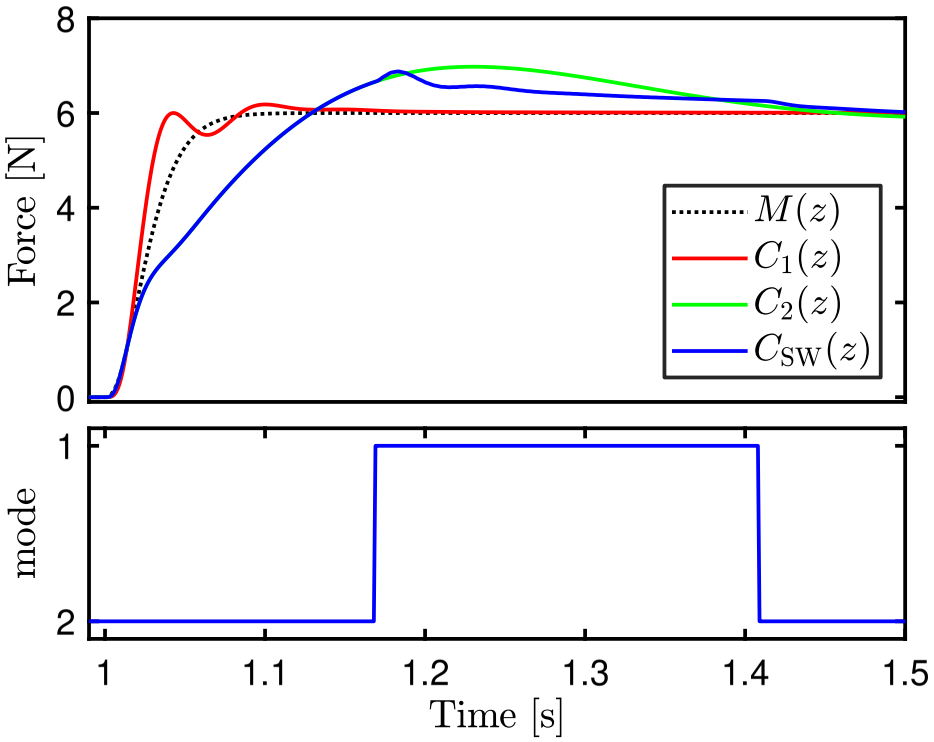
表 6.5 より，適応的制御器を用いることで環境に関わらず所望の特性に平均的に近い応答を実現できることを確認できた。さらに，ゲインスケジュールド制御器を用いた場合には

表 6.5: 平均 2 乗誤差 (MSE) の比較

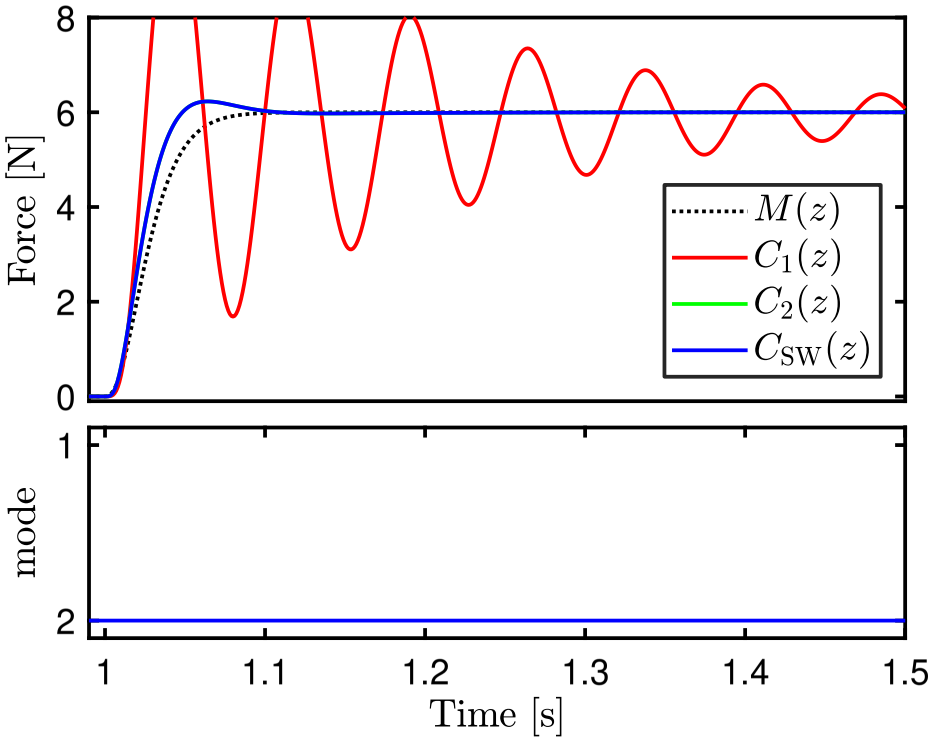
制御器	Z_{e1} に接触	Z_{e2} に接触	平均
$C_1(z)$	4.8271×10^{-2}	1.7278	8.8804×10^{-1}
$C_2(z)$	3.0870×10^{-1}	3.2425×10^{-2}	1.7056×10^{-1}
$C_{\mathrm{SW}}(z)$	2.3631×10^{-1}	3.2425×10^{-2}	1.3437×10^{-1}
$C_{\mathrm{GS}}(z)$	1.8235×10^{-1}	3.4635×10^{-2}	1.0849×10^{-1}

切替型制御器を用いた場合よりも，MSE が小さくできていることが分かる。

以上の結果より，提案手法により力制御においても，環境特性に合わせて制御器を適応させることが可能であることが示された。さらに，切替型制御器をゲインスケジュールド制御器へ拡張することによって，性能向上が見込まれることが確認された。ただし，両方の適応的制御器でステップ応答の印加からモードや可変ゲインが変化するまでの遅れがあることは課題と言える。

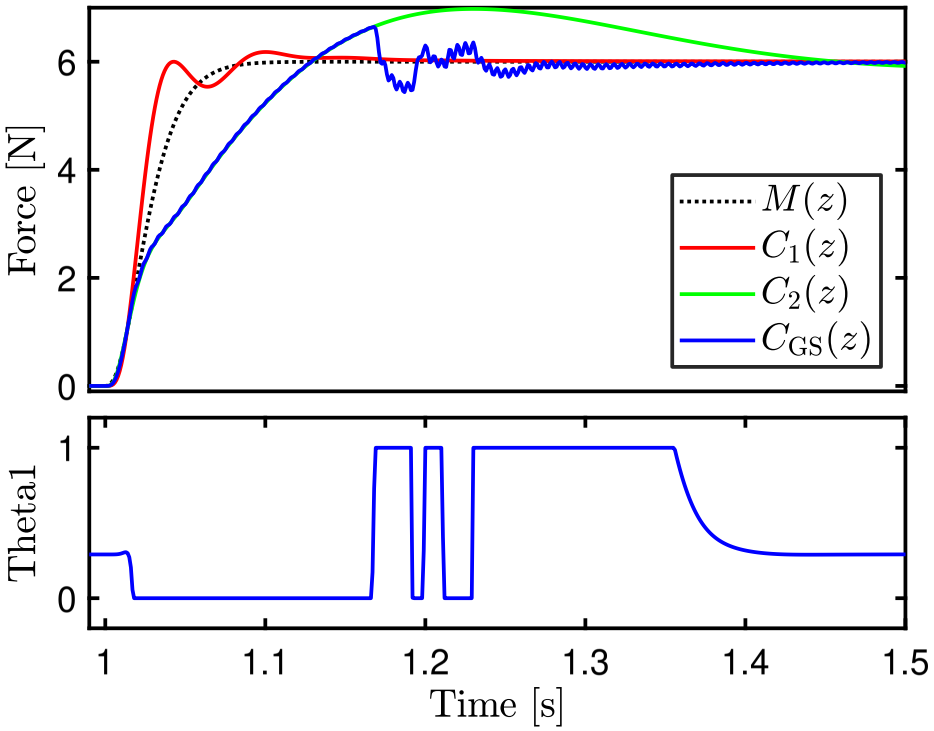


(a) $Z_{e1}(s)$ におけるステップ応答

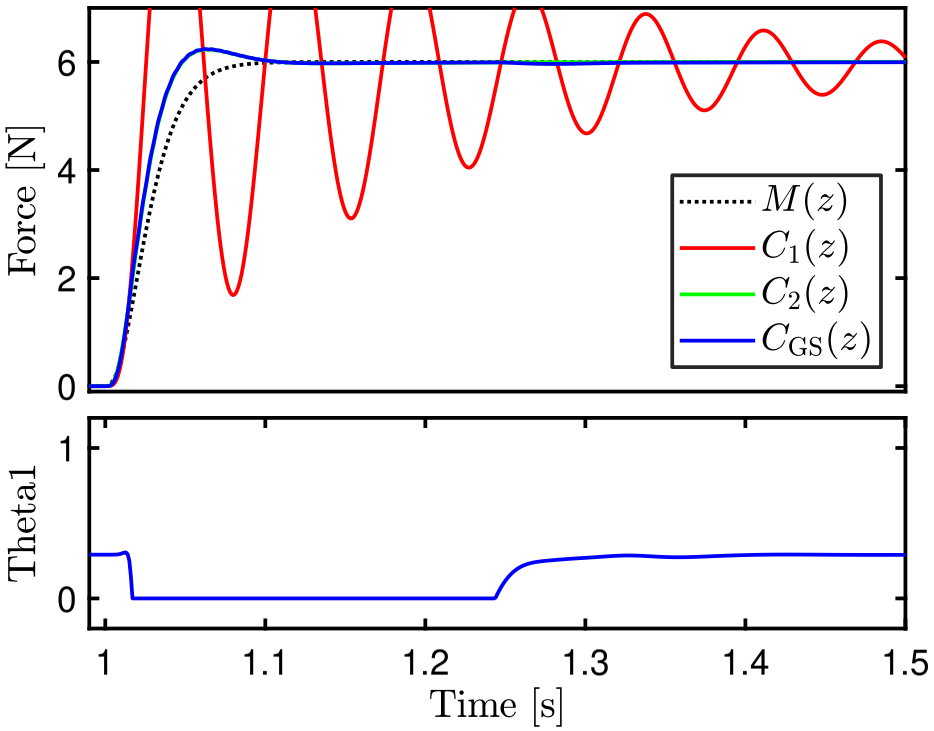


(b) $Z_{e2}(s)$ におけるステップ応答

図 6.4: 切替型制御器によるステップ応答とモード遷移



(a) $Z_{e1}(s)$ におけるステップ応答



(b) $Z_{e2}(s)$ におけるステップ応答

図 6.5: ゲインスケジュールド制御器によるステップ応答とモード遷移

第7章　実機実験

本章では，2 つのスライダを持つリニアモータシステムによる実機実験によって，シミュレーションと同様に提案手法の有効性を検証する。

7.1　実機システム

本節では，実機システムの説明を行う。実機システムは，図 7.1 に示した，1 本のシャフト上に 2 つのスライダを配したデュアルスライダリニアモータを用いる。そして，スライダ 1 では提案する力制御を，スライダ 2 ではインピーダンス制御系を構成した。これにより，インピーダンス制御側の制御パラメータを変更することで，任意の環境特性を模擬して実験が可能となる。

7.1.1　インピーダンス制御

インピーダンス制御[31] とは，ロボットの手先に加わる外力に応じて手先の位置を制御することによって，望みの機械インピーダンスを実現する手法である。

今，図 5.1 におけるインピーダンス特性 $Z_e(s)$ を，リニアモータによるインピーダンス制御で実現することを考える。モータの運動方程式は (7.1) で与えられる。

$$m\ddot{x}(t) = K_t i_e(t) - f_{\mathrm{dis}}(t) \tag{7.1}$$

また，外力 $f_{\mathrm{res}}(t)$ と環境模擬スライダの位置 $x_e(t)$ の関係は (7.3) 式となる。

$$f_{\mathrm{res}}(t) = Z_e(s)\ x_e(t) \tag{7.2}$$

$$= m_e\ddot{x}_e(t) + D_e\dot{x}_e(t) + K_e x_e(t) \tag{7.3}$$

ここで，(7.1) 式と (7.3) 式から $\ddot{x}_e(t)$ を消去しラプラス変換すると (7.4) 式を得る。

$$i_e(t) = \frac{1}{K_{tn}}\frac{m}{m_e}(D_e s + K_e)\left(x_{\mathrm{sur}}(t) - x_e(t)\right) + \frac{1}{K_{tn}}\left(\frac{m}{m_e} - 1\right) f_{\mathrm{res}}(t) \tag{7.4}$$

よって，(7.4) 式を用いてモーター電流指令値を生成することで，望みの環境インピーダンス特性を模擬可能である。

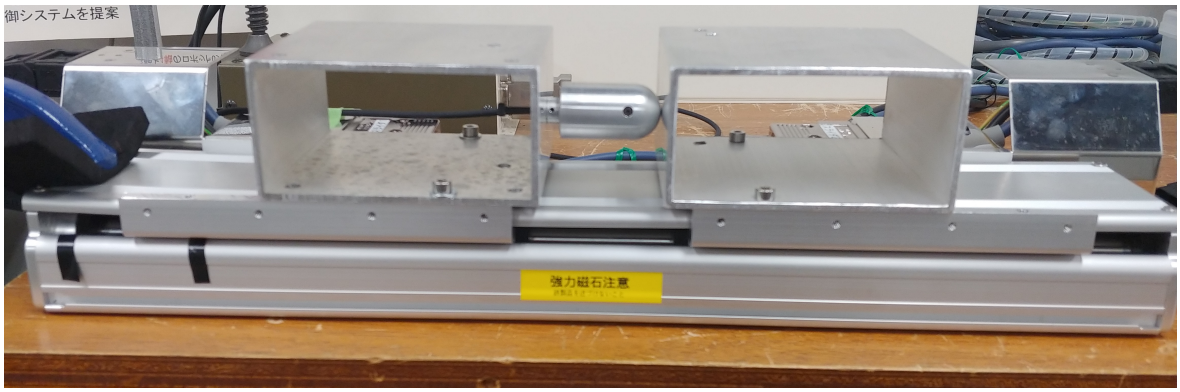


図 7.1: 実験システム

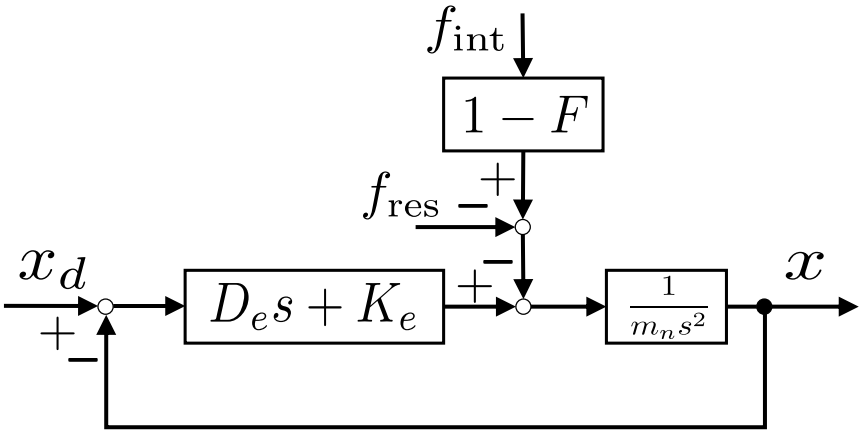


図 7.2: ノミナル化されたインピーダンス制御系

ただし，実機システムにおいては外乱やパラメータ誤差が存在する。そこで，外乱オブザーバを用いて，外乱オブザーバ出力 $d(t)$ から外力を差し引いた信号を補償信号として加えると，補償後のモータ電流指令値 $i_{ea}(t)$ は (7.5) 式で算出できる。

$$i_{ea}(t) = i_e(t) + \frac{1}{K_{tn}} \left(d(t) - f_{\mathrm{res}}(t) \right) \tag{7.5}$$

なお，外乱オブザーバが理想的に働くとき，インピーダンス制御系のブロック線図は図 7.2 となる。ただし，$m_e = m$ と仮定した。

このようにして環境特性をインピーダンス制御で模擬することで，任意の環境特性に対して提案手法の検証を行うことができる。

7.2　実験条件

本節では，実験条件について述べる。また，システムが環境を模擬できているかを確認した予備実験の結果について示す。なお，本章でもシミュレーションと同様に，伝達関数は連続時間伝達関数で記述する。また，設計した適応的制御器は C++ を用いて実装した。なお，行列の計算にはオープンソースの線形代数ライブラリ Eigen[32] を用いた。

実験条件はシミュレーションとなるべく近い条件とした。インピーダンス制御側に与えるインピーダンス特性は (7.6) とする。

$$\begin{cases} Z_{e1}(s) &= 0.75s^2 + 10\sqrt{15}s + 500 \\ Z_{e2}(s) &= 0.75s^2 + 100\sqrt{3}s + 10000 \end{cases} \tag{7.6}$$

次に設計に用いた時系列データについて述べる。入出力データはステップ信号（データ長 $N_1 = 200$）と M 系列信号（データ長 $N_2 = 81915$）を結合したものを用いた。ステップ信号は，エンドエフェクタと環境を接触させた状態でデータを取得させるために印加した。M 系列信号は，振幅 15 mA，5 周期 14 段の M 系列信号を用いた。この信号をそれぞれの環境において印加し，入出力データを取得した。また，それぞれのデータについて，(2.11) 式と (2.8) 式を用いてデータセットを生成した。これら 2 つの環境分のデータを 1 周期ずつ切り出し，結合させたデータ（データ数 $N = 32766$）を設計用のデータセットに用いた。

次に候補制御器の設計条件について述べる。設計する候補制御器の構造を連続時間の伝達関数として，(7.7) 式に示す。

$$\boldsymbol{\beta}(s) = \begin{bmatrix} \beta_4(s) & \beta_3(s) & \beta_2(s) & \beta_1(s) & \beta_0(s) \end{bmatrix}^{\mathrm{T}} \; N(s)\, F_M(s) \tag{7.7}$$

$$\beta_4(s) = \frac{20s}{s+20} \cdot \beta_3(s), \qquad \beta_3(s) = \frac{40s}{s+40} \cdot \beta_2(s),$$
$$\beta_2(s) = \frac{60s}{s+60} \cdot \beta_1(s), \qquad \beta_1(s) = \frac{80s}{s+80} \cdot \beta_0(s), \qquad \beta_0(s) = 1$$

ここで，ノッチフィルタ $N(s)$ は (6.6) 式で与えられる。今回の実験では，シミュレーションと同様に $\delta = 0.5,\ \zeta = 5,\ \omega_n = 1200$ とし，周波数プリワープ処理を行った。また，参照モデル $M(s)$ と周波数重み $W(s)$ は以下で与えた。

$$M(s) = \frac{100}{s+100} \tag{7.8}$$

$$W(s) = \frac{1}{s} \tag{7.9}$$

表 7.1: 実験パラメータ

K_{tn}	33 N/A	推力定数のノミナル値
m_n	0.75 kg	リニアモータのノミナル質量
Z_c	700 kN/m	リニアモータのノミナル質量
g_L	200 rad/s	外乱オブザーバのカットオフ周波数
g_v	400 rad/s	速度推定器のカットオフ周波数
g_f	100 Hz	力センサのカットオフ周波数
T_s	1.0 ms	サンプリングタイム

次にデータ分類について条件を述べる。サブセットの数は $N_S = 762$（サブセットの要素数は $n_s = 43$）とした。また，クラスタリングには Ward 法を用いた。なお，クラスタ数 N_C はデンドログラムの結果を見て決定することとする。

最後に，識別器についての条件を述べる。SVM のカーネル関数には RBF カーネルを用いた。また，ハイパーパラメータであるコストパラメータ η とカーネルパラメータ γ は，試行錯誤的に以下のようにした。

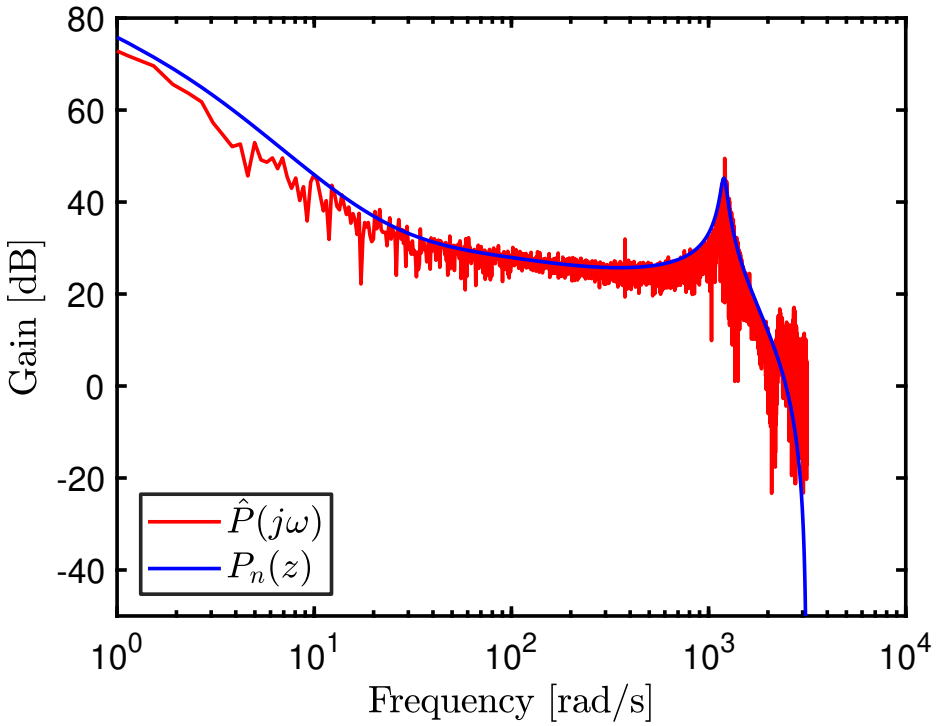
$$\eta = 1000 \tag{7.10}$$
$$\gamma = 0.001 \tag{7.11}$$

以上の条件の下，実験を行った。

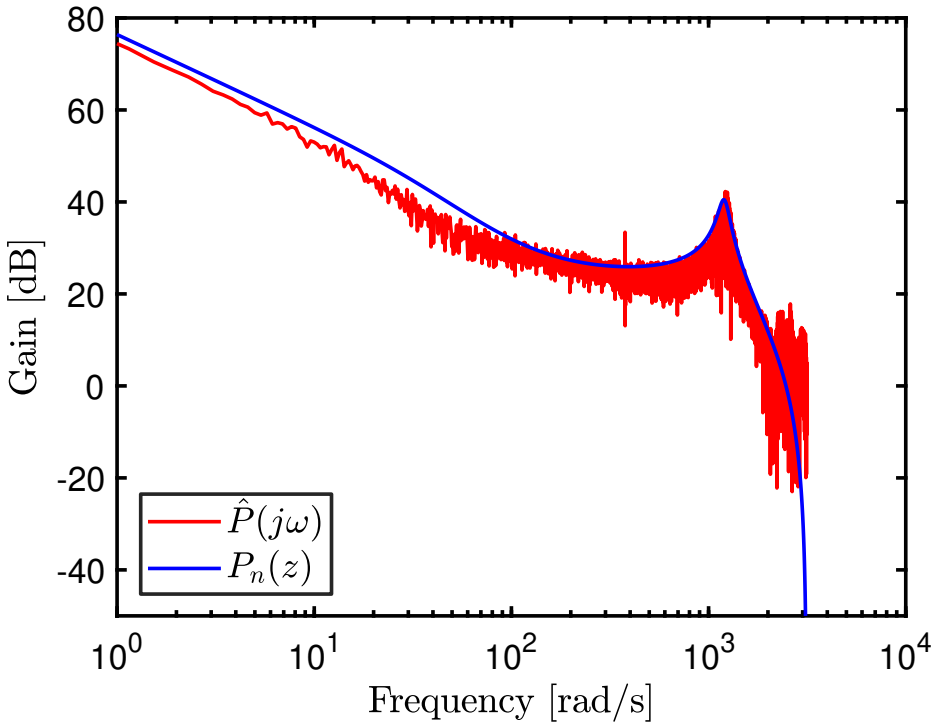
7.2.1　システムの周波数特性の確認

システムが適切にインピーダンス特性を模擬できていることを確認するために，予備実験としてスペクトル解析法による周波数特性の同定実験[33] を行った。力制御側のスライダの電流指令値に M 系列信号を印加し，その時の指令値電流 $i(t)$ から反力 $f_{\mathrm{res}}(t)$ までの周波数伝達関数 $\hat{P}(j\omega)$ を同定した。ただし，入力信号は設計と同様に 14 段 5 周期分の M 系列を信号を印加し，最後の 1 周期を推定に用いた。

同定したゲイン特性を図 7.3 に示す。ここで，ノミナル特性 $P_n(z)$ は (5.10) 式に (7.6) をそれぞれ代入し離散化したものである。図 7.3 より，共振特性も含めて接触時の特性を模擬できていることを確認できた。



(a) $Z_{e1}(z)$ を設定した時



(b) $Z_{e2}(z)$ を設定した時

図 7.3: 実機システムの周波数特性の同定結果

7.3　候補制御器の設計結果

本節では，候補制御器の設計結果について示す。まず，クラスタ数を決めるために，デンドログラムの結果を図 7.4 に示す。デンドログラムより，クラスタ数を 4 から 3 に減らしたとき，一気にクラスタ間距離を大きく確保できることが分かる。しかし，クラスタ数を 3 にするとクラスタ間のデータ数の偏りが大きくなってしまうことも分かる。このようになる理由は，後述するように実験データではパラメータ空間上で外れ値が存在するためであると考えられる。

データ数とクラスタ間距離のバランスを考慮し，今回はクラスタ数 $N_C = 2$ として分類をした結果を図 7.5 に示す。また，シミュレーションと同様にそれぞれのモードに属するデータの数を表 7.2 に示す。この結果より，クラスタのデータ数の偏りが非常に大きくなっていることが分かる。さらに，適合率と再現率を表 7.3 に示す。シミュレーションと比較すると全体的にスコアが下がってしまっていることが分かる。しかし，適合率に注目すれば 50 % 以上のスコアである。従って，それぞれのモードにおいて支配的なデータは，mode1 において Z_{e1} 由来，mode2 において Z_{e2} 由来であることは確認できる。ただし，再現率は非常に差が大きくなってしまったため，識別関数の学習に悪影響があることが示唆される。

次に，前述のパラメータ空間上での外れ値を確認する。ただし，今回の場合パラメータ空間は 5 次元であるため，全ての空間をプロットすることはできない。そこで，ρ_1 と ρ_5 が成す平面に写像を取った場合のパラメータの分類結果を図 7.6 に示す。この分布から，原点付近のデータ群からかなり離れた外れ値が存在することが見て取れる。これらの外れ値の影響によって，クラスタリング結果が偏ってしまったと考えられる。

最後に，分類されたデータから設計した候補制御器のパラメータと評価値を表 7.4 に示す。mode1 の制御器と Z_{e1} 由来データのみから設計したパラメータは比較的近いものとなっていることが分かる。一方で，mode2 の制御器パラメータは Z_{e2} 由来のデータのみから設計したパラメータよりハイゲインな傾向がある。これは，表 7.3 の適合率の低さからも分かるように，mode2 には Z_{e1} 由来のデータが多く混入してしまったためと考えられる。

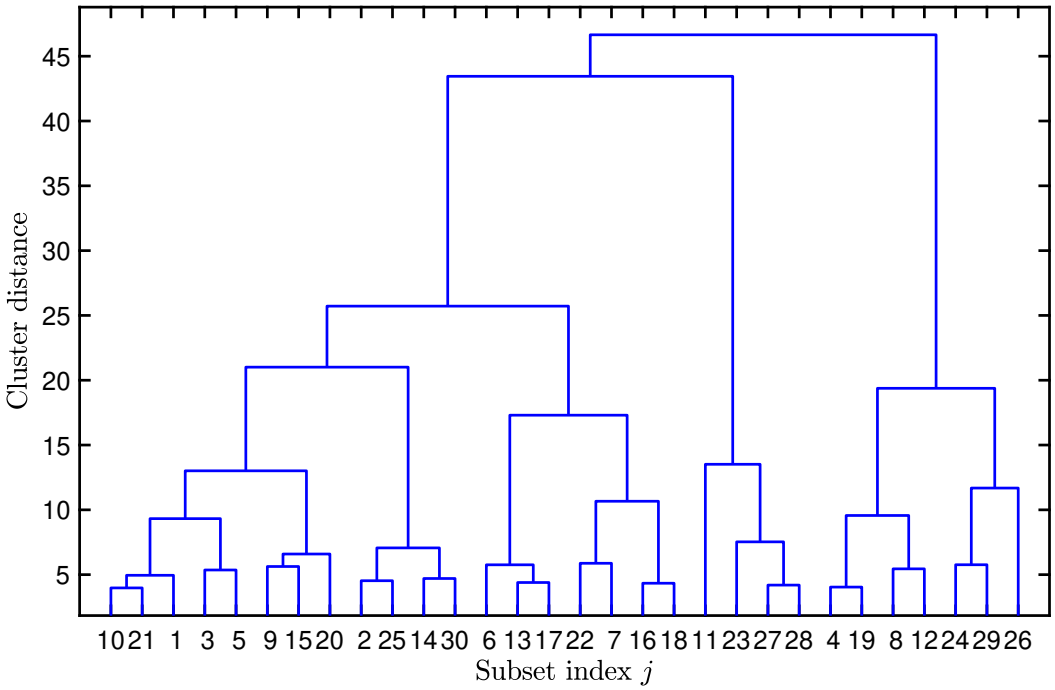


図 7.4: デンドログラム

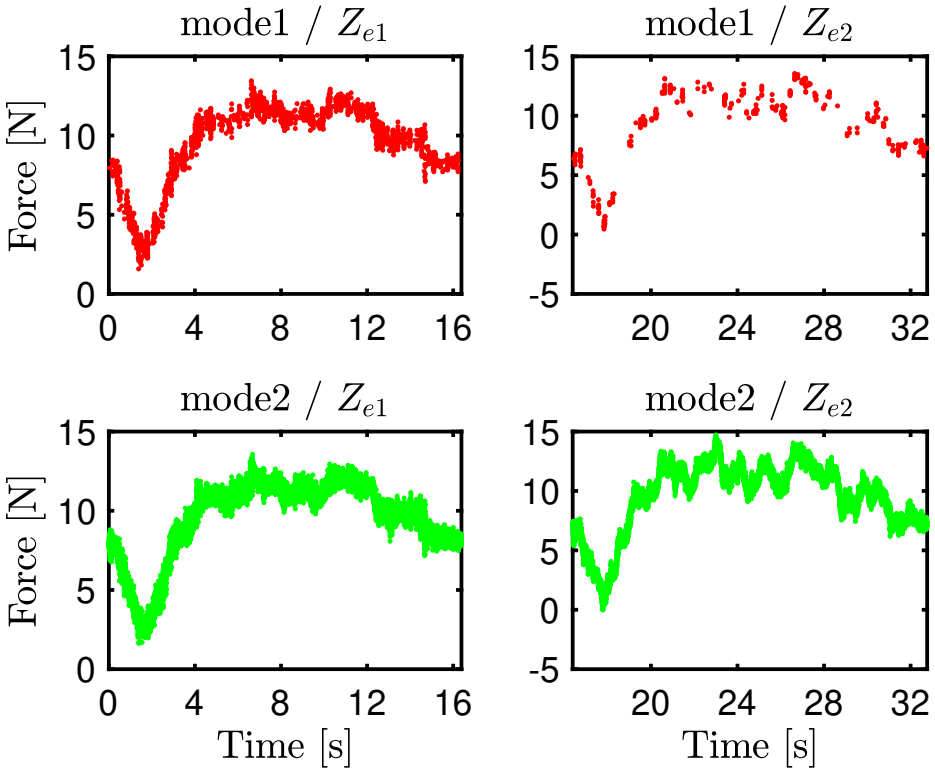


図 7.5: 時系列データの分類結果

表 7.2: 混同行列

	Z_{e1} 由来	Z_{e2} 由来	合計
mode1	2038	327	2365
mode2	14345	16056	30401
合計	16383	16383	32766

表 7.3: 分類の定量的評価

	適合率	再現率	F 値
mode1	86.2 %	12.4 %	21.7 %
mode2	52.8 %	98.0 %	68.7 %

表 7.4: 候補制御器の設計結果

データ元	mode1	mode2	Z_{e1}	Z_{e2}
ρ_1	-1.5593×10^{-7}	-6.6991×10^{-9}	-1.7858×10^{-7}	2.8266×10^{-9}
ρ_2	5.4488×10^{-6}	3.0952×10^{-7}	6.3621×10^{-6}	-4.1006×10^{-8}
ρ_3	-1.2091×10^{-4}	-1.3454×10^{-5}	-1.4420×10^{-4}	-4.4462×10^{-6}
ρ_4	1.5562×10^{-3}	3.3545×10^{-4}	1.8795×10^{-3}	2.3736×10^{-4}
ρ_5	7.7833×10^{-3}	6.2007×10^{-3}	6.7828×10^{-3}	6.1570×10^{-3}
J_{VR}^{N}	9.7258×10^{-10}	5.7540×10^{-10}	4.6068×10^{-10}	1.3625×10^{-10}

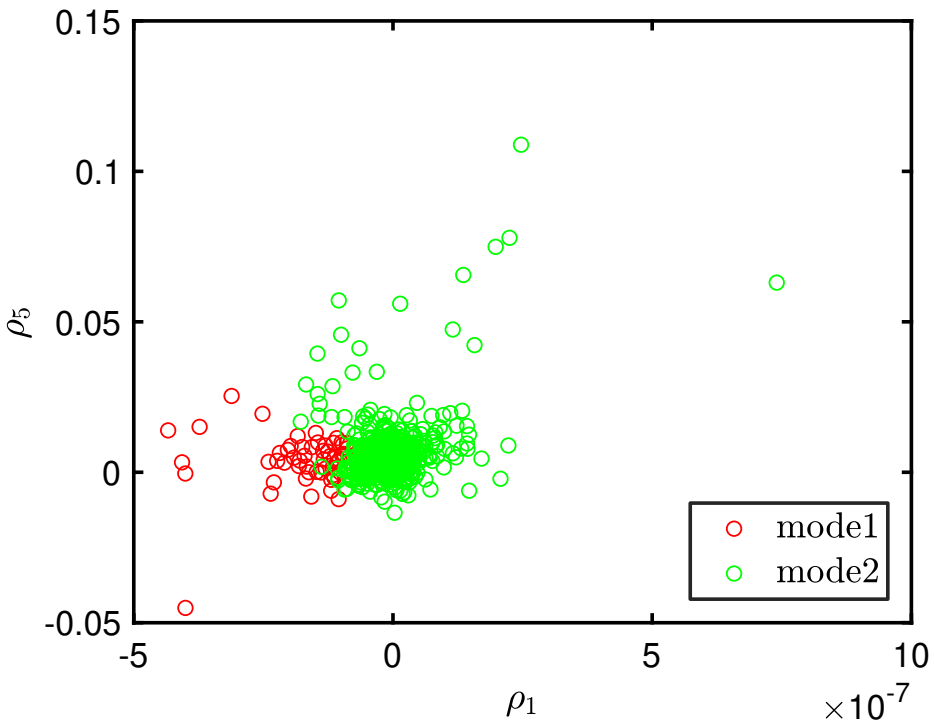


図 7.6: パラメータベクトルの分類結果 (ρ_1 と ρ_5 が成す平面への写像)

表 7.5: 平均 2 乗誤差 (MSE) の比較

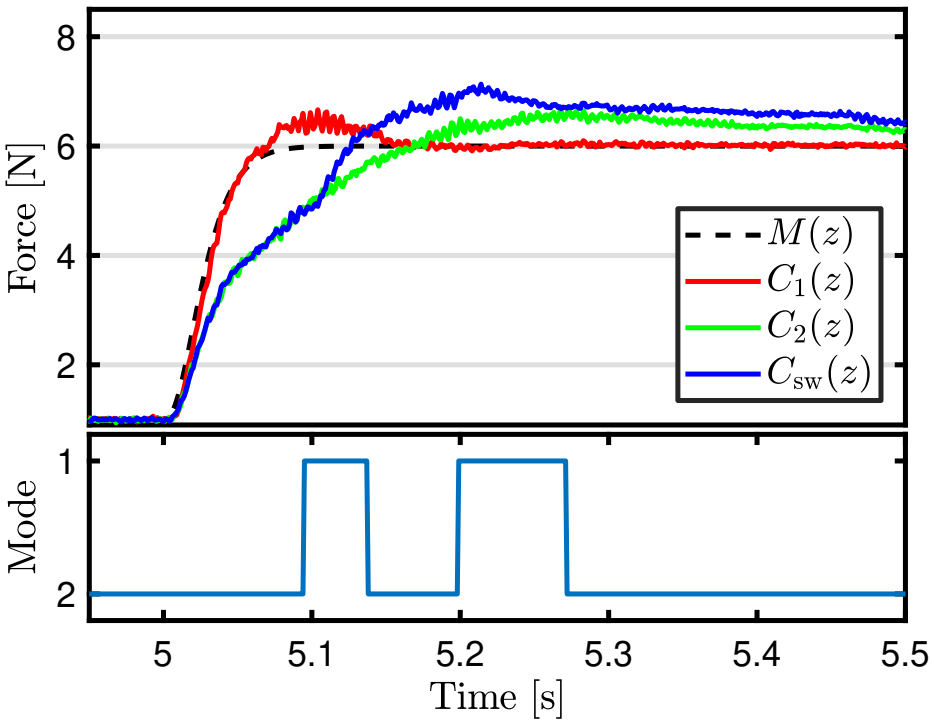
制御器	Z_{e1} に接触	Z_{e2} に接触	平均
$C_1(z)$	1.2826×10^{-3}	6.5726×10^{-2}	6.7009×10^{-2}
$C_2(z)$	1.6863×10^{-2}	8.3582×10^{-3}	2.5221×10^{-2}
$C_{\mathrm{SW}}(z)$	2.4158×10^{-2}	8.1631×10^{-3}	3.2321×10^{-2}
$C_{\mathrm{GS}}(z)$	1.1929×10^{-2}	7.5059×10^{-3}	1.9434×10^{-2}

7.4　適応的制御器の時間応答

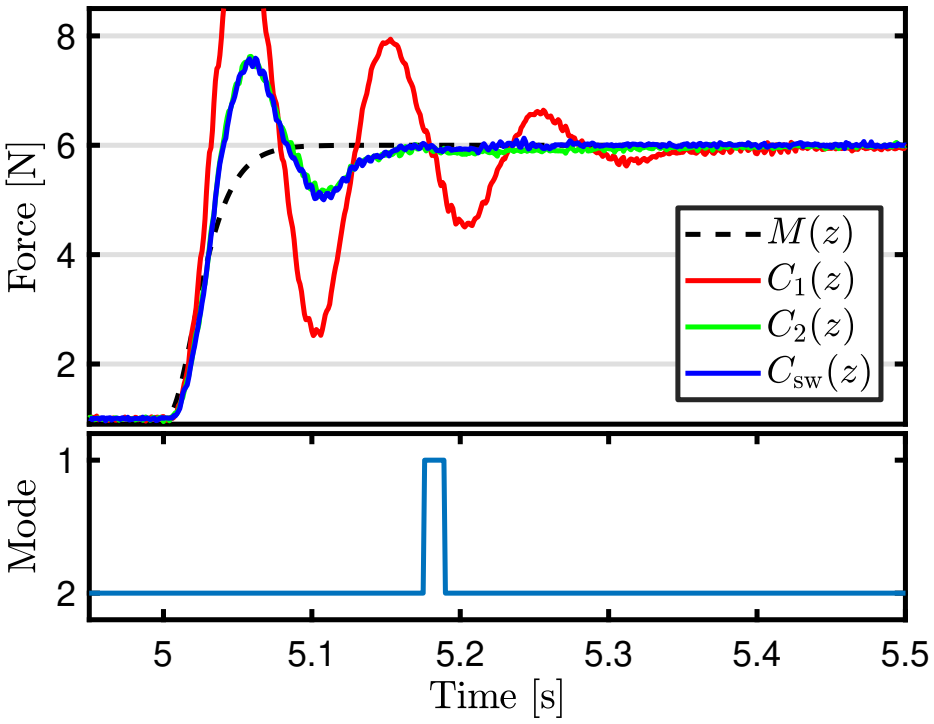
本節では，設計した候補制御器を用いて適応的制御器を実装した時の応答について実機にて検証を行う。ステップ信号は時刻 2.0 s から 5.0 s は 1 N を印加し，5.0 s 以降は 6 N を印加した。このように最初に 1 N を印加しておくことで，確実に環境に接触した状態のステップ応答を取得した。切替型制御器 $C_{\mathrm{SW}}(z)$ のステップ応答を図 7.7 に示す。図 7.7(b) では適切なモード選択を行っているが，図 7.7(a) では切替が適切ではないように思われる。図 7.7(a) では，Z_{e1} に対しては mode1 が適切な制御器モードであるため，時刻 5.1 s 少し前で切替を行っている。しかし，そのタイミングでは $C_1(z)$ の過渡応答はすでにほとんど終わっているため，モードを切り替えても手遅れだったと考えられる。一方で，図 7.8(a) では，ゲインを調節することで整定を早めることができている。さらに，図 7.8(b) でも，$C_{\mathrm{SW}}(z)$ や $C_2(z)$ とほぼ同等の応答を維持している。

また，シミュレーションと同様に MSE を比較した結果を表 7.5 に示す。MSE を比較すると，固定の制御器 $C_2(z)$ を利用した場合の方が切替型制御器よりも平均的に所望の特性に近い。従って，今回のシチュエーションでは切替型制御器は上手く動作しないことが分かる。一方で，ゲインスケジュールド制御を用いた場合には，最も平均的に所望の特性に近い応答を実現している。

シミュレーションと比べて性能が劣化した原因としては，クラスタリングが不正確であることが考えられる。表 7.3 の再現率から分かるように，環境ごとの分類が大きく偏ってしまっていた。そのため，SVM において mode2 の制御器が選ばれる領域が広く学習されてしまったものと考えられる。この問題に対しては，パラメータ空間上での外れ値の除去などの対策が有効だと考えられる。また，そもそも外れ値が発生する原因として力センサのノイズや非線形摩擦の影響が考えられる。そのため，データの前処理による外乱の除去によって外れ値の発生を抑制できる可能性がある。

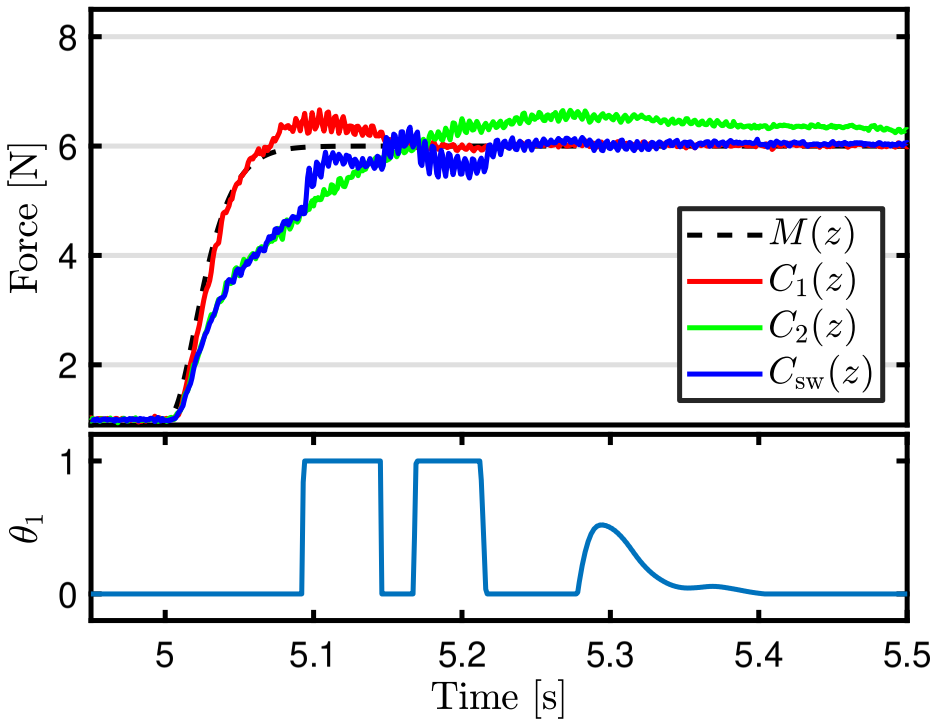


(a) $Z_{e1}(s)$ におけるステップ応答

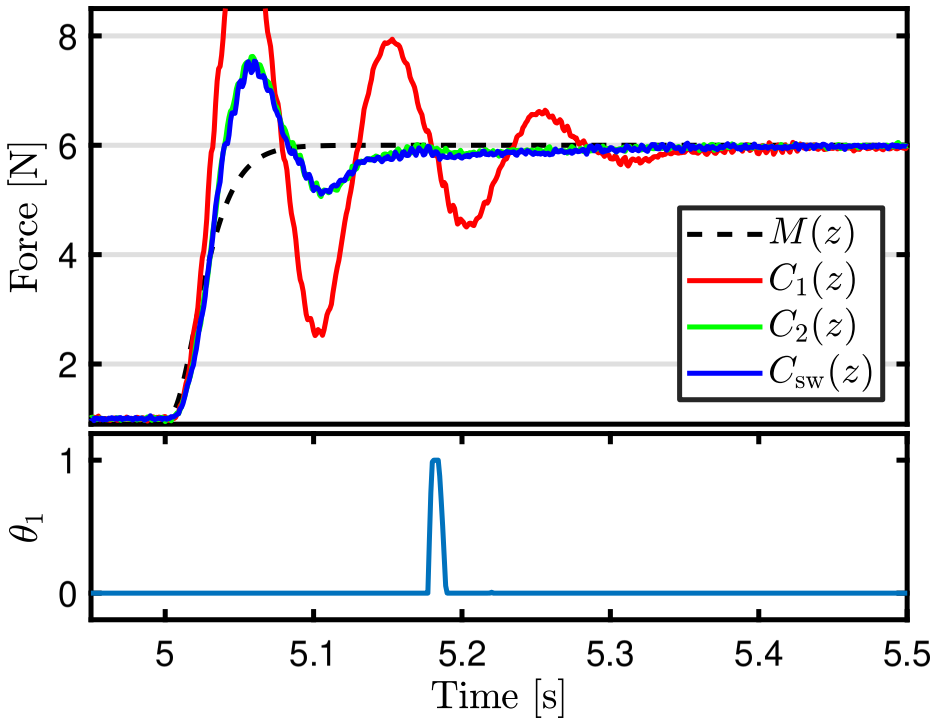


(b) $Z_{e2}(s)$ におけるステップ応答

図 7.7: 切替型制御器によるステップ応答とモード遷移



(a) $Z_{e1}(s)$ におけるステップ応答



(b) $Z_{e2}(s)$ におけるステップ応答

図 7.8: ゲインスケジュールド制御器によるステップ応答とモード遷移

第8章　結言

8.1　まとめ

本研究では，力制御において機械学習を用いて制御器を変化させる適応的制御器の設計法を提案した。力制御では接触対象の動特性が性能や安定性に大きく影響を与える問題があった。そのため，接触対象に合わせて制御器を変化させる適応的制御器に注目した。

適応的制御器の設計法の一つにデータ駆動型制御器設計法と機械学習に基づく切替型制御器設計法が提案されていた。しかし，この手法は制御対象の変化が 1 つの状態量のみに依存する場合についてのみ有効性が確認されていた。これに対して，力制御系では環境の動特性は複数の状態量に依存して変化するため，そのまま適用することはできない。

そこで，本研究では上記の課題を解決するために，モード判別に用いる SVM におけるカーネル関数の変更を提案した。従来手法では多項式カーネルを用いていたが，多項式カーネルでは単純な超平面しか描くことができない。そこで，本研究では RBF カーネルを導入した。これにより，複数の状態量から特性が決まる力制御においても，切替型制御器が有効であることが確認できた。

ただし，切替型制御系では切替時に制御系への悪影響を避けるために，制御器の状態量補償やチャタリング防止のためのヒステリシス特性などを考慮する必要があった。さらに，状態量補償は性能を考慮したものではないため，過渡応答の劣化の可能性もあった。そのため，切替自体を無くし滑らかにモード遷移が行える制御器構造が望ましいと考えた。そこで，切替型制御器のゲインスケジュールド制御器への拡張を提案した。切替型制御器と同様に複数の線形制御器を持つ構造は維持しつつ，これらを切替するのではなく可変ゲインで混合する構造とした。これにより，可変ゲインを滑らかに変化させれば，制御器のモード遷移も滑らかなものとできる。また，可変ゲインの決定には SVM の識別関数の値を用いて決定する方法を考案した。これにより，従来手法の利点を引き継ぎつつも滑らかなモード遷移が可能となった。以上の提案手法について，リニアモータを用いた 1 自由度ロボットにおける力制御系において有効性を検証した。シミュレーションでは有効性が確認できたものの，実機検証では外れ値の影響が大きいことが分かった。

8.2　今後の課題

今後の課題としては以下の 5 つが挙げられる。

1 つ目は，SVM のパラメータ調整が試行錯誤的である点である。機械学習の分野においては，交差検定を用いて正しく分類できた割合を評価関数としたチューニング方法がある。その中でも，基本となる手法はグリッドサーチという手法である。これは，パラメータ探索範囲を与えて，その範囲を全探索する方法である。しかし，この方法では探索に要する時間は膨大になる。そこで，ベイズ推定に基づいて確率モデルを作成し，評価値が良くなる確率が高いパラメータのみを探索させる方法も提案されている[27]。これらの手法を応用することで，SVM のハイパーパラメータについてもオートチューニングが可能となると期待できる。しかし，問題点としてこれらの手法の評価関数が分類の正答率のみである点が挙げられる。経験的にではあるが，本研究において正答率が高いモデルを用いても実際の制御性能は悪い場合が少なからずあった。そのため，機械学習の分野におけるオートチューニングを用いるためには，制御性能を直接評価するような評価関数を検討することが必要であると考えられる。

2 つ目は，階層的クラスタリングや SVM 以外の機械学習の手法を適用することである。本研究においては，データの分類に階層的クラスタリング，データの識別には SVM を用いた。しかし，機械学習の分野においては他にも様々な方法が提案されている。Python における有名な機械学習ライブラリである scikit-learn[34] が公開している比較図を図 8.1 と図 8.2 に示す。これらの手法には得意とするデータ構造がそれぞれ存在するため，通常は問題に合わせて使い分けられる。そして，例えば，この中から DBSCAN とニューラルネットワークを選んで，Ward 法と SVM を置き換えたとしても提案手法による設計は可能である。したがって，他の手法を選択しても設計手順は変更は不要であるが，より高い制御性能を実現できる可能性がある。

3 つ目は，スケジューリングパラメータの決定法についてである。本研究では，SVM の識別関数に基づいて可変ゲインを調整した。そのため，識別関数は緩やかに変化することが望ましい。しかし，SVM はそもそも 2 値分類のための手法であるためマージン領域は狭いほど望ましい。マージン領域が狭いということは，本手法においては滑らかに遷移する領域が狭いということに相当する。そのため，切替制御ではなくゲインスケジュールド制御器においては SVM は必ずしも最適とは言えない。これを解決するためには，確率モデルを用いた識別器を用いることが望ましいと考えられる。つまり，ある制御器モードを用いるべき確率が分かるのであれば，その確率を用いてモード遷移を行う。これによ

り，より滑らかに制御器モードが遷移するのではないかと考えられる。

4 つ目は，安定性の考慮である。本研究においては安定性については考慮していない。しかし，少なくともデータを取得した範囲の制御対象に対して，実装前に安定性を判断できないことは実用上問題となる。本研究で扱った制御器においては，候補制御器が安定化できることとモード遷移しても不安定化しないことが求められる。候補制御器が制御対象を安定化することは，データを用いて小ゲイン定理や受動定理などを適用することで判別が可能であると考えられる。一方で，モード遷移時の安定性の判別は難しいと考えられる。制御対象のモデルが利用可能な場合には，リアプノフの安定条件から安定性を判別可能である[35]。しかし，データ駆動型制御器設計のフレームワークにおいては利用できないため，別のアプローチを考える必要があるだろう。

5 つ目は，他の制御対象への適用が可能であるかを検討することである。本研究では，1 自由度の力制御系を対象として設計法の検証を行った。しかし，提案手法は機械学習に基づくため理論的な検証は難しく，本紙でも数値例による有効性の検証のみを行った。そのため，提案手法が一般に適用可能であることを検証するには，より多くの違う状況での検証が必要である。例えば，ショベルカーのような物体を運ぶ 2 自由度シリアルリンクロボットシステムは物体の質量や姿勢によって動特性が変化する。このような，他の非線形システムにおいても適用可能であれば，汎用的な設計法であると言えるだろう。

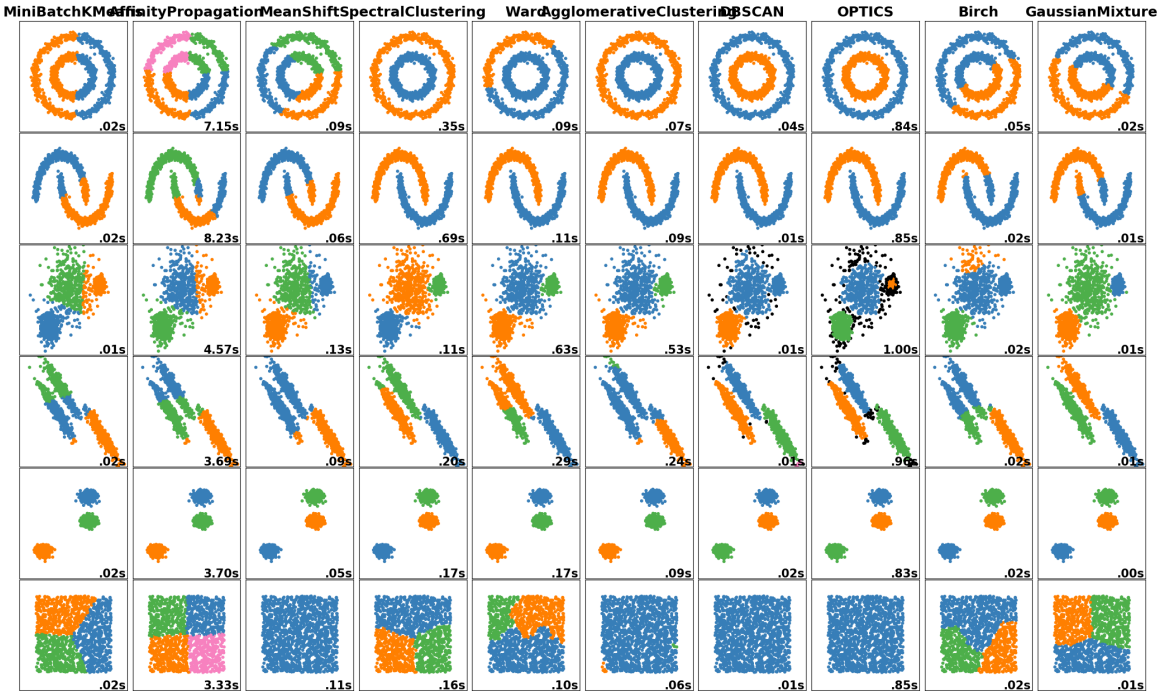


図 8.1: クラスタリングの比較

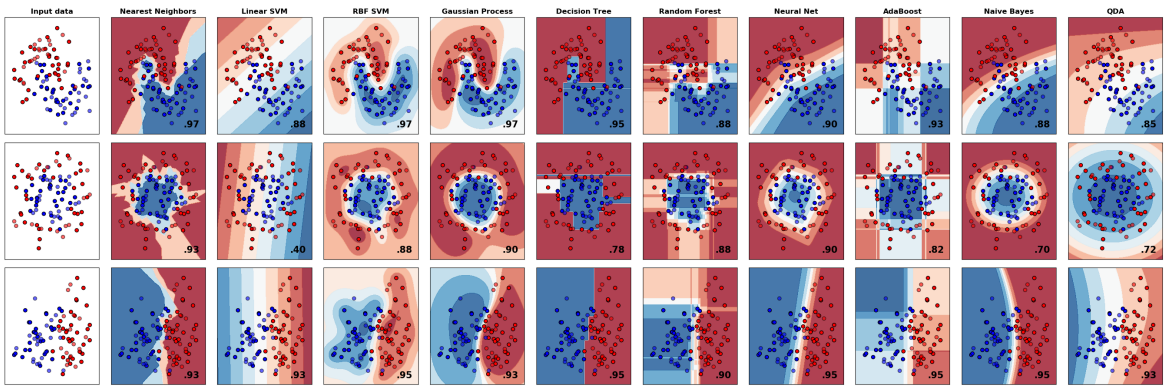


図 8.2: 識別器の比較

参考文献

[1] 榊原伸介,「ロボット技術, IoT および AI の活用による製造業の競争力強化」, 精密工学会誌, vol. 83, no. 1, pp. 30–35, (2017)

[2] N. Hogan, "Impedance control: An approach to manipulation: Part I – Part III", *Journal of dynamic systems, measurement, and control*, vol. 107, no. 1, pp. 1–24, Mar. 1985.

[3] E. Magrini, F. Flacco, and A. De Luca, "Control of generalized contact motion and force in physical human-robot interaction", in *2015 IEEE international conference on robotics and automation (ICRA)*, 2015, pp. 2298–2304, 2015.

[4] M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators", *Journal of Dynamic Systems, Measurement, and Control*, vol. 103, no. 2, pp. 126–133, 1981.

[5] 三上知三,「精密仕上げロボットシステムを用いた実用化例と今後の展開」, 日本ロボット学会誌, vol. 34, no. 10, pp. 676–679, (2016)

[6] 志村康治, 堀洋一,「ロボットマニピュレータにおける力制御のロバスト化と衝突過程の制御」, 日本ロボット学会誌, vol. 11, no. 2, pp. 235–245, (1993)

[7] 村上俊之, 大西公平,「作業環境のパラメータ推定に基づいた多自由度マニピュレータの力制御」, 電気学会論文誌 D (産業応用部門誌), vol. 113, no. 4, pp. 503–509, (1993)

[8] 鵜田正俊, 福田敏男,「神経回路モデルによるロボットマニピュレータの力制御 (追加学習型ニューラルネットワークを用いた一自由度マニピュレータの適応力制御)」, 日本ロボット学会誌, vol. 14, no. 1, pp. 75–82, (1996)

[9] K. Burn, M. Short, and R. Bicker, "Adaptive and nonlinear fuzzy force control techniques applied to robots operating in uncertain environments", *Journal of robotic systems*, vol. 20, no. 7, pp. 391–400, 2003.

[10] 山本茂,「ハイブリッドシステムの制御-VI: 同定手法」, システム/制御/情報, vol. 52, no. 3, pp. 103–109, (2008)

[11] 丸田一郎,「区分的アフィンシステムの同定 (< 特集> システム同定と推定の最近の動向)」, システム/制御/情報, vol. 59, no. 5, pp. 180–185, (2015)

[12] 田中秀幸,「LPV システムの同定手法 (< 特集> システム同定と推定の最近の動向)」, システム/制御/情報, vol. 59, no. 5, pp. 174–179, (2015)

[13] M. C. Campi and S. M. Savaresi, "Direct nonlinear control design: the virtual reference feedback tuning (VRFT) approach", *IEEE Transactions on Automatic Control*, vol. 51, no. 1, pp. 14–27, 2006.

[14] M. Sakamoto, T. Hamaguchi, Y. Ota, and Y. Hashimoto, "Nonlinear control design using vrft", *Journal of chemical engineering of Japan*, vol. 40, no. 10, pp. 832–839, 2007.

[15] 薮井 将太, 弓場井 一裕, 平井 淳之,「1 リンク鉛直型アームに対するモデルフリー制御器設計法によるスイッチング制御器の直接設計」, SICE 三重地区計測制御研究講演会講演論文集, pp. 1–4, (2008.12)

[16] 弓場井一裕 山本直輝, 矢代大祐, 駒田諭,「非反証制御に基づく切り替え型力制御器の提案」, 電気学会メカトロニクス制御研究会資料, pp. 29–32, (2017)

[17] S. Wakitani, T. Yamamoto, and B. Gopaluni, "Design and Application of a Database-Driven PID Controller with Data-Driven Updating Algorithm", *Industrial & Engineering Chemistry Research*, 2019.

[18] M. C. Campi, A. Lecchini, and S. M. Savaresi, "Virtual reference feedback tuning: a direct method for the design of feedback controllers", *Automatica*, vol. 38, no. 8, pp. 1337–1346, 2002.

[19] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari, "A clustering technique for the identification of piecewise affine systems", in *International Workshop on Hybrid Systems: Computation and Control*, 2001, pp. 218–231, 2001.

[20] 藤田 政之, 平田 研二,「スイッチング制御」, 計測と制御, vol. 38, no. 3, pp. 176–181, (1999)

[21] A. B. Arehart and W. A. Wolovich, "Bumpless switching in hybrid systems", in *International Hybrid Systems Workshop*, 1996, pp. 1–17, 1996.

[22] 山口 高司, 宍田 和久, 平井 洋武,「磁気ディスク装置ヘッド位置決めサーボ系におけるサーボモード切り換え時の初期値補償問題の基礎検討」, 計測自動制御学会論文集, vol. 29, no. 7, pp. 792–799, (1993)

[23] 神嶌敏弘,「データマイニング分野のクラスタリング手法 (1): クラスタリングを使ってみよう!」人工知能学会誌, vol. 18, no. 1, pp. 59–65, (2003)

[24] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines", *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 27:1–27:27, 3 2011, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[25] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding", in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035, 2007.

[26] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification", 2003.

[27] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms", in *Advances in neural information processing systems*, 2012, pp. 2951–2959, 2012.

[28] 薮井 将太, 弓場井 一裕, 平井 淳之,「HDD のヘッド位置決め制御におけるモデルフリー制御器設計法によるスイッチング制御器の直接設計」, 産業計測制御研究会論文集, pp. 99–104, (2009)

[29] 大西公平,「外乱オブザーバによるロバスト・モーションコントロール」, 日本ロボット学会誌, vol. 11, no. 4, pp. 6–13, (1993)

[30] 山口 高司, 平田 光男, 藤本 博志,「ナノスケールサーボ制御: 高速・高精度に位置を決める技術」. 東京電機大学出版局 (2007)

[31] 川崎 晴久,「ロボット工学の基礎」. 森北出版 (1991)

[32] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[33] 足立 修一,「システム同定の基礎」. 東京電機大学出版局 (2009)

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] 増渕 正美, 川田 誠一,「システムのモデリングと非線形制御」. コロナ社 (1996)

謝辞

本研究の遂行ならびに修士論文作成にあたって，熱心なご指導とご鞭撻を賜りました，三重大学大学院工学研究科教授 駒田 諭 先生，同大学准教授 弓場井 一裕 先生，同大学准教授 山村 直紀 先生，同大学助教 矢代 大祐 先生，同大学助教 小山 昌人 先生に心から感謝の意を表します。

弓場井 一裕先生には，グループ検討会や学会発表練習など多くの場面で厳しくも温かさを感じるを指導していただき，非常に感謝しております。何かと至らない点も多い自分にも実機作成や学会発表の機会を与えてくださって，そのおかげで自分も成長できたように思います。この場をお借りして心よりお礼申し上げます。

矢代 大祐先生には，本研究の実機実験に関して多くの貴重な助言を頂きました。実機作成は初めての経験でしたが，先生に時間を取って指導して頂いたおかげで，実機作成と実験をやり遂げることができました。

また，実機作成時の部品購入を始め，日頃から研究室生活でお世話になった技術職員 中村 勝 氏に心から感謝します。

本研究を遂行するにあたり，同じ制御理論グループの先輩として親身なご指導を頂きました，山本 直輝 氏，宇佐美 朋大 氏，堀 智貴 氏，岡村 雅行 氏，伊達 宗充 氏，柳川 晃佑 氏に深く感謝いたします。そして同じ制御理論グループの同期として協力し，切磋琢磨し合いながら研究を遂行してきた堀田 敦 君，森田 晃史 君に深く感謝します。また，何かと至らない自分を受け入れ親身になって交流して頂いた，酒徳 大雅 氏を始めとした制御理論グループとロボットグループの皆さまに心から感謝します。

最後に，遠く離れていても研究生活を支援してくださった両親に心から感謝します。

論文目録

国際会議

(1) S. Tsujii, K. Yubai, D. Yashiro, S. Komada: “A Study on Design Method of the Switching System for Force Control”, *Proceeding of 12th France–Japan and 10th Europe–Asia Congress on Mecatronics*, JD–001325, pp. 119–124 (2018.9)

(2) S. Tsujii, K. Yubai, D. Yashiro, S. Komada: “A Fundamental Study on Switching Control System with Multiple Modes for Force Control”, *Proceedings of the 5th IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization (2019)*, TT8–6, pp. 1–6 (2019.3)

(3) S. Tsujii, K. Yubai, D. Yashiro, S. Komada: “A Fundamental Study on Switching Control System with Multiple Modes for Force Control”, *Proceedings of the 6th IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization (SAMCOM2020)* (2020.3 発表予定)

学会発表

(1) 辻井祥太郎，弓場井一裕，矢代大祐，駒田諭：「データ駆動型制御器設計法に基づいた切替型力制御器の設計法」，平成 30 年度電気・電子・情報関係学会東海支部連合大会講演論文集，I4–1 (2018.9)

(2) 辻井祥太郎，弓場井一裕，矢代大祐，駒田諭：「データ駆動型制御器設計法と機械学習に基づいた適応的力制御器の設計法の提案」，電気学会制御研究会資料，CT–20–010, pp.47–52 (2020.1)

付録 A　実機システムについて

本章では，提案手法の有効性を検証するために作成した実機システムについて説明を行う。

A.1　ハードウェア仕様

本節ではハードウェアについて説明する。ハードウェア全体の構成は，図 A.1 に示すとおりである。本システムは，構造可変型ロボットと共有の Linux PC を用いて実時間制御を行う。そして，Linux PC のインターフェイスボードから制御盤を通してをサーボドライバと信号のやり取りを行う。また，SW-Box は 2 つのスライダを個別にサーボオンできるようになっている。これらについて，以下で説明していく。

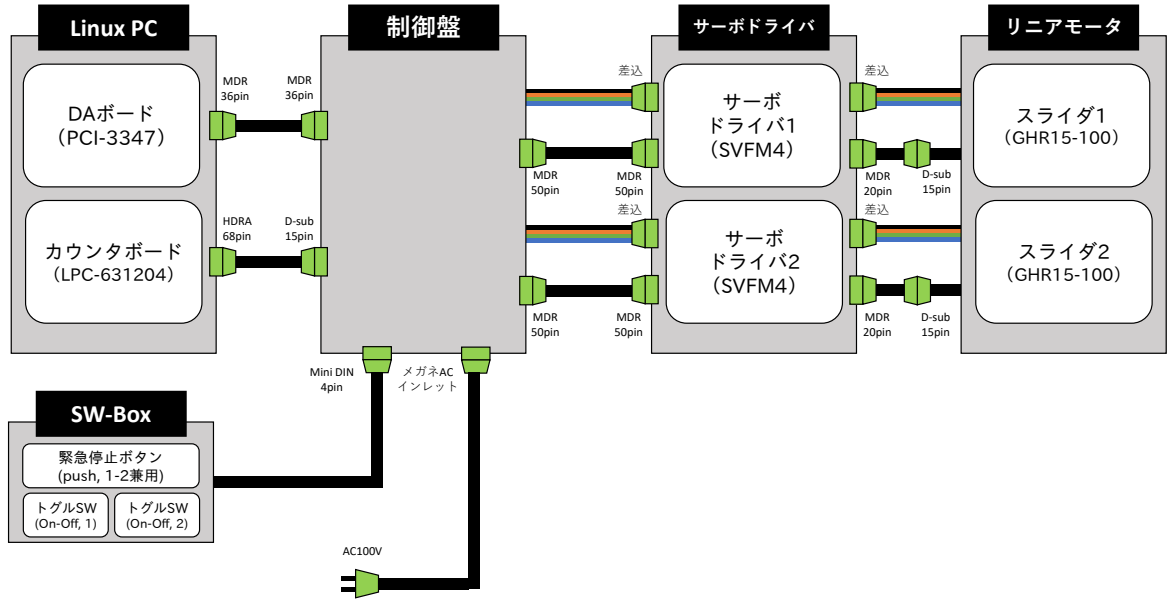


図 A.1: ハードウェア全体の構成

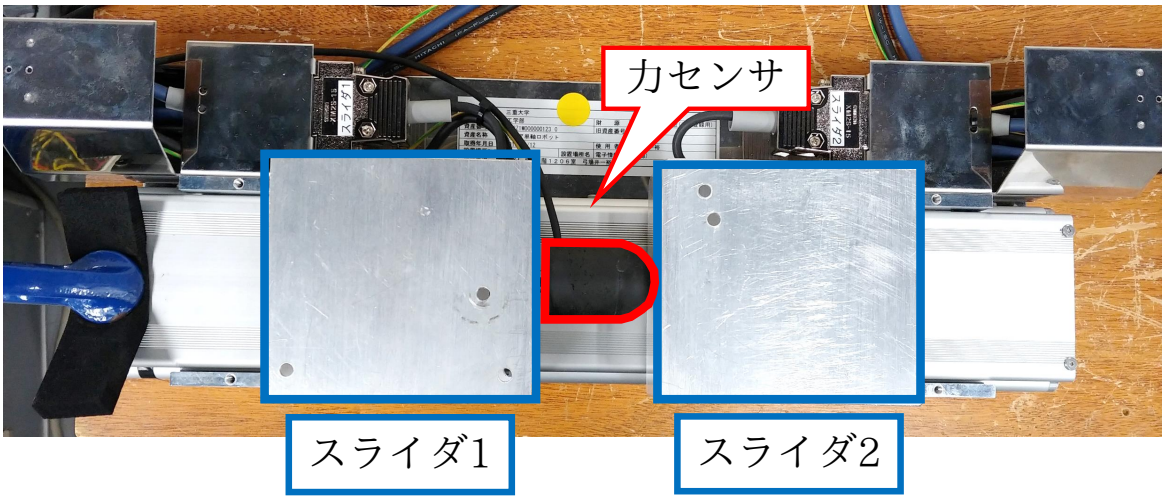


図 A.2: リニアモータの外観図

A.1.1　リニアモータ

リニアモータはエンコーダ付きのダブルスライダタイプの単軸ロボットを購入した。リニアモータ本体の外観を図 A.2 に示す。リニアモータとエンコーダの仕様を表 A.1 と表 A.2 に示す。

本研究では力センサを用いて力制御を行うために力センサをスライダ上に固定する必要があった。そのため，アルミの角パイプを切断し，パイプを介してスライダ上に固定している。なお，アルミ部材の重量は約 0.21 kg である。

A.1.2　サーボドライバ

サーボドライバはリニアモータの説明書内で推奨されていた汎用サーボドライバを購入した。サーボドライバの仕様を表 A.3 に示す。

サーボドライバには設定が必要なパラメータがいくつか存在するが，RS232C 経由でパラメータの設定が可能である。RS232C での通信用のケーブルは，PC 側が D-Sub9，サーボドライバ側が RJ11 の特殊なケーブルである。自作した通信用ケーブルの外観を図 A.3 に示す。表 A.4 には設定が必要なサーボドライバのパラメータと変更後の値を記す。それぞれのパラメータの詳細については，ドライバの説明書を参照されたい。

表 A.1: リニアモータの仕様

メーカー	ジイエムシーヒルストン株式会社
型番	GHR15-100-D-M3-A3-MH
ストローク長	100 mm
定格推力	17 N
定格電流	0.51 A
推力定数	33 N/A
スライダ質量	0.54 kg
リニアブロック数	2 個
リニアブロックシール抵抗	2 N/個

表 A.2: エンコーダの仕様

メーカー	ハイデンハイン株式会社
型番	LIDA279
エンコーダタイプ	インクリメンタルリニアエンコーダ
分解能	0.001 mm(4 逓倍)
電源	5 V

表 A.3: サーボドライバの仕様

メーカー	サーボランド株式会社
型番	SVFM4-H3-X
主電源	AC100 V
制御対象	3 相ブラシレス永久磁石同期型サーボモータ
制御方式	ソフトウェアサーボ + 電流ループ + 転流ループ
サーボ周期	8 kHz
通信機能	RS232C 経由の MOV/2 コマンド
連続定格電圧 (RMS)	59.4 V
連続定格電流 (RMS)	2.5 A
重量	1 kg
絶対最大電圧	AC 142 V, DC 200 V

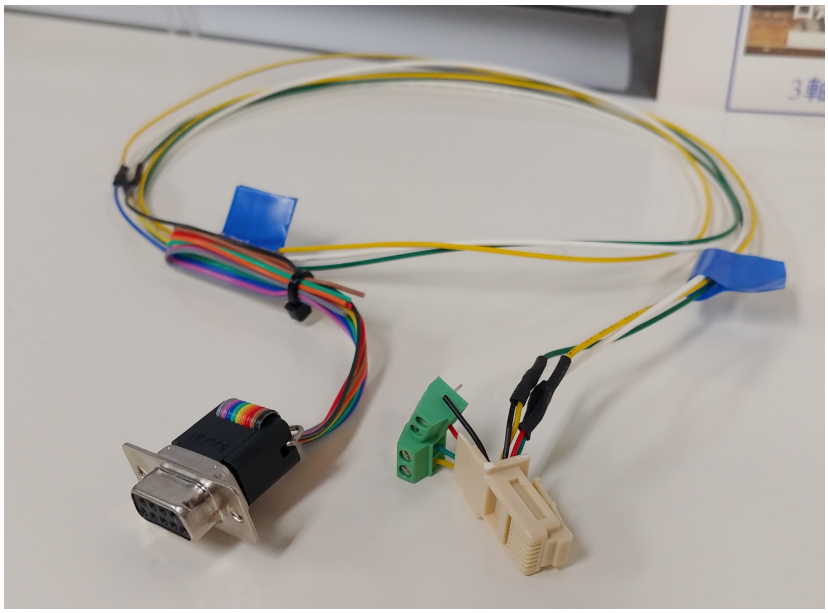


図 A.3: 通信用ケーブル (D-Sub9 – RJ11)

表 A.4: ドライバのパラメータ設定

No.	名前	設定値	単位	補足
#000	エンコーダ基本分解能	15000	pulse ／基準 mm	計算式は説明書 p.101 にある
#001	リニアモータ基準長さ	60	mm	リニアの仕様書より
#002	ポール数	2		N-N ワンセットなら 2
#003	モータ定格電流	5	0.1 A(rms)	モータ定格 x10 を四捨五入
#004	電流リミット値	150	%	加速トルク以下で安全な値へ
#007	最大回転数	3000	mm/s	リニアの仕様書より
#008	検出モード	145		エンコーダの相の回転方向に注意
#011	I ／ O 形式	1		アナログ入力形式
#012	初期モード	37		トルク電流制御
#015	電子ボリューム	3125		5V で 1Arms(200%) 出力となる値
#016	電子トリマ	測定値	pulse	指令値 0V の時の ANIN1 の平均値
#032	スイッチ	0		再現性のため，ITU を自動実行しない
#038	モータタイプ	102		AC 型リニアモータで設定

A.1.3　制御盤とスイッチボックス

制御盤はノイズ対策も兼ねて金属製のものを購入した。制御盤には，AC ノイズフィルタ，スイッチング電源，配線用の端子台，サーボドライバが収められている。制御盤内部の配置図を図 A.4 に示す。ここで，後ろに数字が付いているものは，対応するスライダを制御するために使われている。例えば，スライダ 1 は端子台 1 とドライバ 1 のみに依存して制御される。

端子台 1 の結線図は，図 A.5 に示すとおりである。端子台 2 も SW-Box の電源に関する配線以外は同様である。図 A.5 から分かるように，スイッチボックスの SV1 は SV-ON に結線されている。したがって，スイッチボックスからサーボオンの制御ができるようになっている。また，DA ボードやカウンタボードとの信号の受け渡しはシングルエンドとなっている。ただし，シングルエンド信号はノイズの影響が重畳しやすいと言われている。そのため，ノイズが問題になるような場合には差動信号への変更を検討する必要がある。

スイッチボックスの外観と内部の配線の対応関係を A.6 に示す。SW1 と SW2 をオンにすることで，スライダ 1 とスライダ 2 をそれぞれ独立にサーボオンできる。SW3 には LED が内蔵されており，SW3 の LED が点灯中のみ SW1 と SW2 は有効となる。つまり，SW3 は SW1 と SW2 の上位のスイッチであるため，暴走時には SW3 だけをオフすれば全てのモータを停止できる。

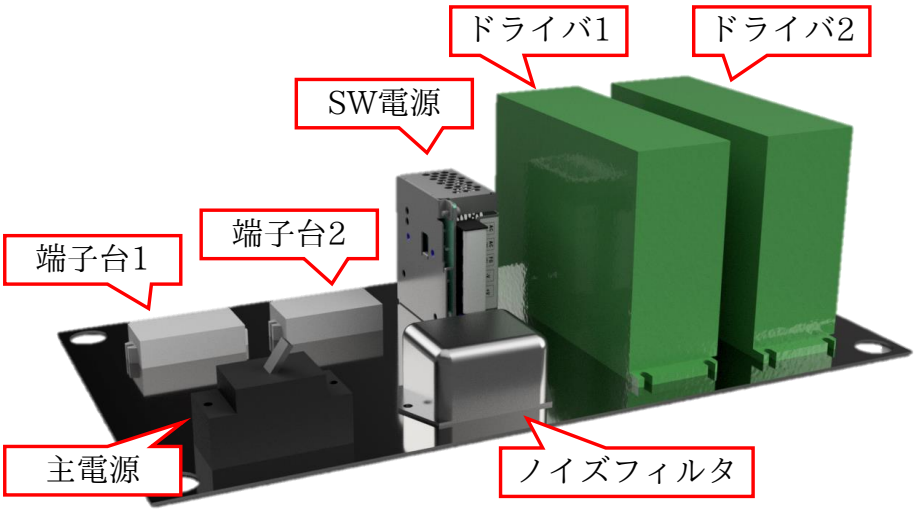


図 A.4: 制御盤の内部配置

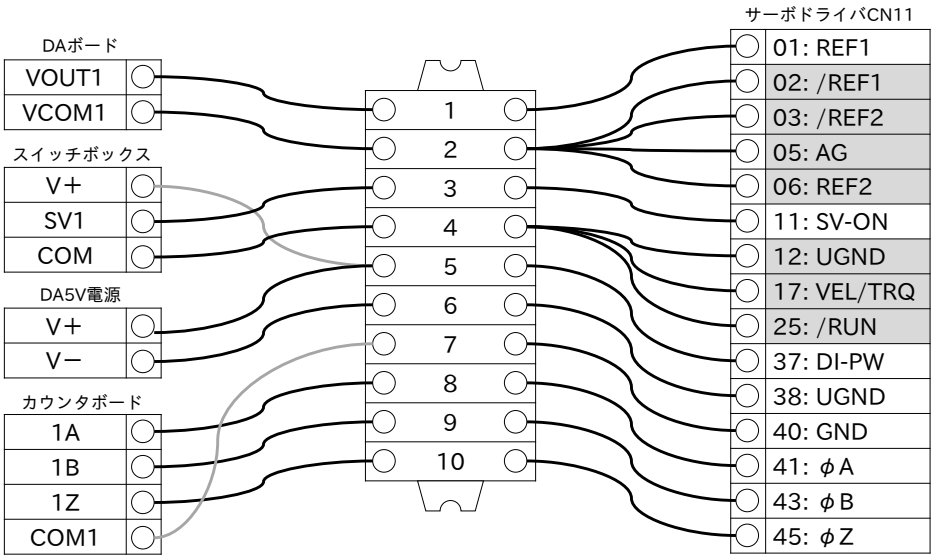
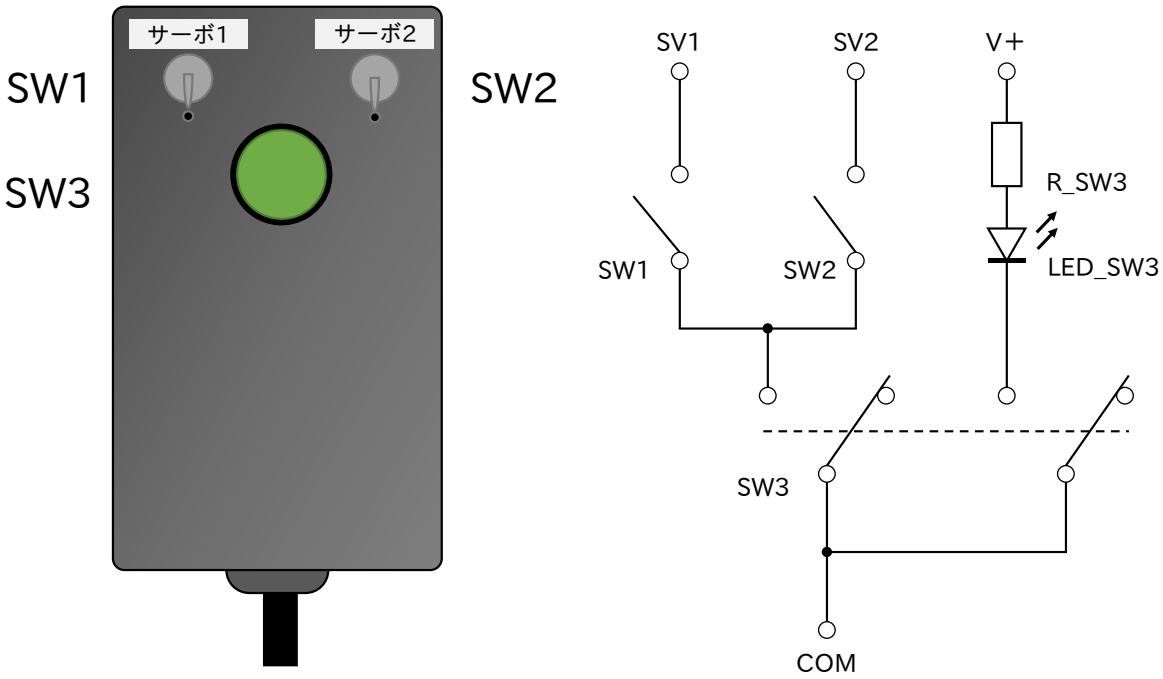


図 A.5: 端子台 1 の結線図



(a) 外観図　　(b) 結線図

図 A.6: スイッチボックスの仕様

A.1.4　構造可変ロボットと共用する部分

制御用の PC とインターフェイスボード，および力センサユニットは構造可変型ロボットと共用している。表 A.5 に実機 PC，表 A.6，表 A.7，表 A.8 にインターフェイスボードの仕様を示す。また，表 A.9 に力センサユニットの仕様を示す。力センサユニットの校正行列は，表 A.10，表 A.11 の通りである。なお，2020 年 1 月時点で，シリアル番号が NA0439 の力センサについては劣化が激しく，センサ値が正常に出ない可能性が高い。よって，可能な限りシリアル番号が NA0440 のセンサを用いて実験を行うことが望ましい。

表 A.5: 制御用の Linux PC の仕様

CPU	Core2Duo 2.80GHz (E7400)
メモリ	2 GB
ストレージ 1	261 GB (構造可変用)
ストレージ 2	50 GB (リニアモータ用)

表 A.6: 万能ボード (AD 分のみ記載)

メーカー	株式会社インターフェイス
型番	PCI-360116
分解能 (AD)	16 bit
チャンネル数 (AD)	16
入力電圧 (AD)	バイポーラ $\pm 1.25, \pm 2.5, \pm 5, \pm 10$V
最高サンプリング速度 (AD)	1MSPS

表 A.7: DA ボード

メーカー	株式会社インターフェイス
型番	PCI-3347
分解能	16 bit
チャンネル数 (DA)	4
電圧レンジ	バイポーラ $\pm 5, \pm 10$V
変換時間	10 µs(電圧出力時)

表 A.8: カウンタボード

メーカー	株式会社インターフェイス
型番	LPC-631204
カウンタ	32 bit
チャンネル数	4
最高入力周波数	2 MHz

表 A.9: 力センサユニット

メーカー	ビー・エル・オートテック株式会社
型番	NANO1.2/1-A
シリアル番号	NA0439, NA0440
定格荷重（力）	11.8 N
定格荷重（トルク）	9.8×10^{-2} N m
増幅器カットオフ	100 Hz

表 A.10: 校正行列（シリアル番号：NA0439）

-0.004528	-0.044789	0.042923	0.854473	-0.828074	-0.812537	0	0
-0.141836	-0.893371	0.009103	0.440495	0.08408	0.485752	0	0
-1.233387	-0.017562	-1.228226	-0.036266	-1.22064	-0.056078	0	0
-0.013127	-0.059413	-0.777754	0.000023	0.704381	0.058104	0	0
0.93041	0.034691	-0.410179	-0.181845	-0.461794	0.114801	0	0
-0.016597	-0.313931	-0.039944	-0.387272	-0.051408	-0.257727	0	0

表 A.11: 校正行列（シリアル番号：NA0440）

0.061961	0.065228	0.063811	0.706524	-0.00766	-0.793869	0	0
-0.037144	-0.853877	0.041732	0.490855	-0.034593	0.375532	0	0
-1.204164	0.256209	-1.152233	-0.156518	-1.344956	-0.131034	0	0
-0.018066	0.52954	-0.457353	-0.308049	0.5197	-0.22397	0	0
0.903692	0.054388	-0.482239	-0.172983	-0.409079	0.095638	0	0
0.013912	-0.327126	-0.058981	-0.354297	0.007127	-0.263163	0	0

表 A.12: リアルタイム OS の仕様

ディストリビューション	Ubuntu 16.04 LTS
ディストリビューションのサポート期間	2021 年 4 月まで
カーネルバージョン	4.9.80
RTAI のバージョン	5.1

A.2　ソフトウェア仕様

本節では，Linux PC のソフトウェアについて説明する。

A.2.1　リアルタイム OS

Linux OS を用いてリアルタイム制御を行うために，本システムでは RTAI (`https://www.rtai.org/`) を用いている。この RTAI は既存の Linux カーネルへのパッチの形式で配布されており，パッチを当ててカーネルを再コンパイルすることで利用できる。RTAI を適用した OS の仕様を表 A.12 に示す。表 A.12 にあるサポート終了までは，apt コマンドなども通常の Ubuntu と同様に使用できる。

実際に実時間制御を行うプログラムを書くためには RTAI の API を呼び出す必要がある。簡単な実時間制御を行うサンプルプログラムを Listing A.1 と Listing A.2 に示す。このサンプルプログラムは制御周期 1.0×10^5 ns で，任意の電圧を出力する。出力結果は，1.0×10^8 ns ごとに標準出力に表示され，最後に全てのデータがファイルに書き出される。ただし，77 行目の “`controller(&daVoltage);`“の controller 関数は外部で実装されているものとする。

API についてのより詳細な情報は，公式のドキュメントを参照されたい。

Listing A.1: rtai_test.hpp

```
1  #ifndef RTAI_MAIN
2  #define RTAI_MAIN
3  #include <rtai.h>    //To use rtai functions
4  #include <rtai_msg.h> //To use rtai utils
5  #include <pthread.h> //To create threads
6  #include <fbida.h>   //To control DA board
7  #include <fbipenc.h> //To control Counter board
8  #include "controller.hpp"
9
10 /*
11  * Constant values
12  */
13
14 /* SAMPLING_TIME - sampling time [ns]*/
15 constexpr int SAMPLING_TIME = 1e+5;
16 /* PERIODIC_ITE_MAX - Maximum iteration of periodic timer */
17 constexpr int PERIODIC_ITE_MAX = 1e+5;
```

```
18  /* PRINTING_TIME - print data rate [ns] */
19  constexpr int PRINTING_TIME = 1e+8;
20  /* USED_DA_CH_NUM - the nunber of ch updated at the same time.*/
21  constexpr int DA_USED_CH_NUM = 1;
22  /* DA_DEVICE_NO */
23  constexpr int DA_DEVICE_NO = 1;
24  /* FILE_NAME_TO_SAVE - */
25  constexpr char FILE_NAME_TO_SAVE[] = "result.dat\0";
26  /* DA_MAX_VOLT */
27  constexpr double DA_MAX_VOLT = 5;
28  /* DA_MIN_VOLT */
29  constexpr double DA_MIN_VOLT = -5;
30  /* DA_RESOLUTION */
31  constexpr unsigned short DA_RESOLUTION = 0xffff;
32  /* DA_BIAS */
33  constexpr unsigned short DA_BIAS = 0x8000;
34
35
36  /*
37   * structs
38   */
39  /* shm_recorded_datas - online data strage to print and to save. */
40  struct shm_recorded_datas
41  {
42      int idx;
43      int real_periodtime_ns;
44      double da_output_voltage;
45      double test;
46  };
47
48  /*
49   * Functions
50   */
51
52  /* thread1 - manages periodic timer */
53  void *thread_control_core( void* arg );
54
55  /* thread_print - print data periodic */
56  void *thread_print( void* arg );
57
58  /* main - called first*/
59  int main(int argc, char *argv[]);
60
61  // extern void controller(unsigned short *DA_data, int idx);
62  #endif
```

Listing A.2: rtai_test.cpp

```
1
2   /* Header files */
3   #include "rtai_test.hpp"
4
5   /* Global variables */
6   pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
7   bool has_been_finished_control = false;
8   struct shm_recorded_datas shm_datas[PERIODIC_ITE_MAX];
9
10  /* this defines finish time */
11  /* ここで指定した回数で終了します(PREIODIC_ITE_MAX以下で有効) */
12  #define NUM_TEST 20000
13
14  void *thread_control_core( void* arg )
15  {
16      void* result = NULL;
17      RT_TASK *task1 = NULL;
18      int nRet;
19      RTIME t_base;
20      RTIME t_now;
21      RTIME t_diff;
22      double daVoltage;
23      unsigned short daDataHex[DA_USED_CH_NUM];
24      DASMPLCHREQ DaSmplChReq[DA_USED_CH_NUM];
25      DaSmplChReq[0].ulChNo = 1;
26      DaSmplChReq[0].ulRange = DA_5V;
27
28      /* Create new realtime task */
29      task1 = rt_task_init_schmod(
30          nam2num("MYTHD1"), /* ID */
31          0,  /* Priority */
32          1024,  /* Stack size */
```

```
33          1024,  /* Max message size */
34          SCHED_FIFO, /* Task scheduling policy */
35          0x0f   /* Processor affinity */
36          );
37      if( NULL == task1 )
38      {
39          printf("Failed to create task1\n");
40          return( result );
41      }
42
43      // DA board opening
44      nRet = DaOpen(DA_DEVICE_NO);
45      if (nRet != DA_ERROR_SUCCESS)
46      {
47          printf("Open failed(0x%x)\n", nRet);
48          return ( result );
49      } else {
50          rtai_print_to_screen("DA Open is succeeded");
51      }
52      DaSetOutputDAEx(DA_DEVICE_NO, DA_USED_CH_NUM, &DaSmplChReq[0]);
53
54      /* Make this task realtime */
55      rt_make_hard_real_time();
56
57      /* do realtime job */
58      /* Timer is started and became periodic */
59      start_rt_timer( 0 );
60      rt_task_make_periodic_relative_ns(
61          task1, /*this task*/
62          0, /*start delay [ns]*/
63          SAMPLING_TIME /*period time [ns]*/
64      );
65
66      // t_base = rt_get_time_ns();
67      for(int i = 0; i < PERIODIC_ITE_MAX; i++){
68          /* Periodic block*/
69          /* DO NOT USE PRINTF() IN THIS BLOCK */
70          /* if you wannna print, use shm_print_datas. */
71          /* このfor文が実時間制御部分です。printfなど重い処理は行わないこと。*/
72          rt_task_wait_period();
73          t_now = rt_get_time_ns();
74
75          /* Calucuration */
76          /* この関数はファイルを分けていますので、そちらを編集してください */
77          controller(&daVoltage, i);
78
79          /* DA borad output */
80          if(daVoltage > DA_MAX_VOLT){daVoltage = DA_MAX_VOLT;} // 電圧リミット
81          if(daVoltage < DA_MIN_VOLT){daVoltage = DA_MIN_VOLT;} // 電圧リミット
82          daDataHex[0] = daVoltage/(DA_MAX_VOLT - DA_MIN_VOLT)*DA_RESOLUTION + DA_BIAS;
                // 16進変換
83          nRet = DaOutputDAEx(DA_DEVICE_NO, &daDataHex[0]); // DA出力
84          if( nRet != DA_ERROR_SUCCESS)
85          {
86              // DA出力の失敗時は、強制的に制御を終了する
87              rtai_print_to_screen("DA output is failed."); // dmesgのバッファへ出力される
88              puts("DA output is failed.");
89              pthread_mutex_lock(&mymutex);
90              has_been_finished_control = true;
91              pthread_mutex_unlock(&mymutex);
92          }
93
94          /* Others */
95          if(i == NUM_TEST) break; // NUM_TEST で強制的に終了させる
96          t_diff = t_now - t_base;
97          pthread_mutex_trylock(&mymutex);
98          shm_datas[i].idx = i;
99          shm_datas[i].da_output_voltage = daVoltage;
100         shm_datas[i].real_periodtime_ns = t_diff;
101         pthread_mutex_unlock(&mymutex);
102         t_base = t_now;
103     }
104
105     // DA board closing
106     nRet = DaClose(DA_DEVICE_NO);
107     if(nRet == DA_ERROR_SUCCESS)
108     {
109         rtai_print_to_screen("DA close is succeeded."); // dmesgのバッファへ出力される
110     }
111
112     // RtTimer is stopped
113     stop_rt_timer();
114
```

```
115         /* Delete realtime task */
116         rt_task_delete( task1 );
117
118         /* finish flag is changed*/
119         pthread_mutex_lock(&mymutex);
120         has_been_finished_control = true;
121         pthread_mutex_unlock(&mymutex);
122         /* Exit thread */
123         return( result );
124     }
125
126     void *thread_print(void* arg)
127         {
128         void* result = NULL;
129         RT_TASK *task2 = NULL;
130         struct shm_recorded_datas localData;
131
132         /* create new realtime task */
133         task2 = rt_task_init_schmod(
134             nam2num("MYTHD2"),
135             10,
136             1024,
137             1024,
138             SCHED_FIFO,
139             0x0f
140         );
141         if(NULL == task2)
142         {
143             puts("Failed to create thread_print");
144             return result;
145         }
146
147         /* Make this task realtime */
148         rt_make_soft_real_time();
149
150         rt_task_make_periodic_relative_ns(
151             task2, /* This task */
152             0,     /* Start delay[ns] */
153             PRINTING_TIME /* period [ns] */
154         );
155
156         for(int i = 0; i < PERIODIC_ITE_MAX; i++)
157         {
158             /* Periodic block */
159             rt_task_wait_period();
160             pthread_mutex_lock(&mymutex);
161             localData = shm_datas[i];
162             if (has_been_finished_control)
163             {
164                 /*if cocntrol is finished, printing should finish too*/
165                 pthread_mutex_unlock(&mymutex);
166                 break;
167             }
168             pthread_mutex_unlock(&mymutex);
169             if(localData.idx == 0) { continue; }
170             printf(
171                 "%05d (dT = %7.03f [us]) : %+06.3f\n",
172                 localData.idx,
173                 ( localData.real_periodtime_ns/1000.0 ),
174                 localData.da_output_voltage
175             );
176         }
177
178         /* Delete realtime task */
179         rt_task_delete( task2 );
180         /* Exit thread */
181         return( result );
182     }
183
184     int main(int argc, char *argv[])
185     {
186         pthread_t th1;
187         pthread_t th2;
188         RT_TASK *mainTask = NULL;
189         FILE *resultFile;
190         time_t timer;
191         struct tm *date;
192         char timeStamp[32];
193         char actualFileName[256];
194
195         /* Enable non-root hrt */
196         rt_allow_nonroot_hrt();
197
```

```
198         /* Create realtime task */
199         mainTask = rt_task_init_schmod(
200                nam2num("MAINTH"),/* ID */
201                10, /* Priority */
202                1024, /* Stack size */
203                1024, /* Max message size */
204                SCHED_FIFO, /* Task scheduling policy */
205                0x0f /* Processor affinity */
206            );
207         if( NULL == mainTask )
208         {
209             printf("Failed to create main task.\n");
210             return( 0 );
211         }
212 
213         /* Create user thread */
214         pthread_create( &th1, NULL, thread_control_core, NULL );
215         pthread_create( &th2, NULL, thread_print, NULL );
216 
217         pthread_join( th1, NULL );
218         pthread_join( th2, NULL );
219 
220         /* write down the result */
221         timer = time(NULL);
222         date = localtime(&timer);
223         strftime(timeStamp, 255, "./results/%Y%m%d_%H%M%S\0", date);
224         snprintf(actualFileName, 255, "%s_%s", timeStamp, FILE_NAME_TO_SAVE);
225         if((resultFile = fopen(actualFileName, "w")) == NULL)
226         {
227             puts("file open error.");
228             return 1;
229         }
230         pthread_mutex_lock(&mymutex);
231         for(int i = 0; i < NUM_TEST; i++)
232         {
233             fprintf(resultFile,
234                 "%05d\t%9.4f\t%+07.3f\n",
235                 i,
236                 i*SAMPLING_TIME*1.0e-9,
237                 (shm_datas[i].da_output_voltage)
238             );
239         }
240         pthread_mutex_unlock(&mymutex);
241         fclose(resultFile);
242 
243         /* Delete MAINTH */
244         rt_task_delete( mainTask );
245 
246         return 0;
247     }
```

A.2.2　サーボドライバとの通信方法

前述のとおり，サーボドライバとの通信には RS232C によるシリアル通信が使用可能である。そこで，制御用 PC にはシリアル通信用の GUI アプリケーションをインストールしている。シリアル通信用のアプリケーションは，Windows では teraterm が有名であるが，Ubuntu では使えない。そこで，本システムには gtkterm をインストール済みである。初期設定は済ませてあるので，使用の際にはケーブルの接続とアプリケーションの立ち上げのみで通信可能である。なお，Configuration から CRLF auto というオプションにチェックをいれると表示が見やすくなる。また，オンボードではなく USB タイプの RS232C 変換ケーブルを利用する際には Load configuration から”usb“というプロファイルを読み込めばよい。

A.3　実験手順

本節では，実験を行う際の手順について述べる。

手順 1 制御ボックスの主電源とスイッチボックスのスイッチが全てオフであることを確認し，PC の電源をオンする。

手順 2 ブートしたい OS の選択画面（grub）はそのままエンターを押す。
(構造可変用の ART-Linux を起動する場合は，別の項目を選択する)

手順 3 ユーザ名「syotaro」でログインし，端末を起動させる。

手順 4 Desktop へ移動し，”my_setup.sh“というファイルを以下のコマンドで実行する。

```
1   sudo bash my_setup.sh
```

これにより，インターフェイスボード用のデバイスノードの作成と RTAI 用のカーネルモジュールのロードが行われる。

手順 5 ソースコードのあるディレクトリへ移動し，make コマンドを実行する。(sudo は不要) これにより，ソースコードのコンパイルが自動的に行われる。万が一，エラーが表示された場合には，メッセージに従って修正を行うこと。

手順 6 実行時エラーがないか確認するために，生成された実行ファイルを実行する。

```
1   ./run
```

実行ができていることを確認出来たら Ctrl + C で一旦プログラムを終了させる。

手順 7 スライダの位置を手で少し動かし，スライダの前後に動くためのスペースを作る。また，制御盤の主電源をオンし，SW3 の LED を点灯させる。

手順 8 SW1 と SW2 を順にサーボオンし，自動力率検知が終了するのを待つ。

手順 9 力制御を行う場合は，手動で環境とスライダが接触する位置にスライダを移動させる。位置制御を行う場合は，使用しないスライダを端に寄せておく。

手順 10 確認済みの実行ファイルを実行し，実験を行う。この際，暴走に備えてスイッチボックスの SW3 は常に押せるようにしておくこと。

手順 11 実行が終了したら，スイッチボックスのスイッチを全てオフにする。再度実験を行う場合は，**手順 5** からやり直す。