修 士 論 文

ロボットアームの適応制御における計算量の軽減に関する研究

指導教官　示村悦二郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

ブディ　ラフマント
Budi Rachmanto

1997 年 2 月 14 日

要 旨

本稿では, 多入力多出力系のための多変数単純適応制御における計算量の問題を明確にし, それを克服するための対策を提案する. 考案したコントローラの簡略化手法をシミュレーションによって検討し, その結果の性能評価を行う.

また, 単純適応制御手法をロボットマニピュレータのアーム制御に適用した時の起こり得る軸間干渉や非線形性など, 様々な問題点を明確にし, それを克服するための方法を考える.

さらに, 考案した簡略型制御手法を用いて, 超多関節ロボットの超多自由度アーム制御に挑戦を試み, 機構解析プログラム DADS によるシミュレーションの結果を報告する.

目 次

図 目 次

表 目 次

1 序論

現在の産業界では，多種多様な用途において様々な種類の多関節ロボットが幅広く使用されている．ロボット作業の多様化により，これらの多関節ロボットの制御において，今後さらなる高精度化，高速度化が強く要求されており，その実現のためより一層の研究が期待されている．

従来の制御法すなわち, 実プラントのモデルを推定し, プラントモデルのパラメータ同定を行い, 制御理論に基づいて導出したフィードバック係数によるフィードバック制御のような制御方法では, プラントの数式モデルの精度に大きく左右される．また, 汎用性も乏しく, 不確かさが存在するような制御に最適であるとはいえない．

そこでこの問題を克服するため, プラントが未知でも制御可能である適応制御が考案され始めるようになった．適応制御とは制御対象の特性の変化に応じてオンラインで制御則を自動的に変化させ, 高い制御性能を維持する制御手法であり, 代表的な適応制御としてモデル規範形適応制御 MRACS (Model Reference Adaptive Control System) および自己チューニング適応制御法 STR (Self-Tuning Adaptive Regulator) がある．特にモデル規範型制御である MRACS が長年にわたって脚光を浴びていた．MRACS とは, 特性未知のプラントを制御するにあたって, プラントとコントローラを一体とした制御系の特性が規範モデルと呼ばれる理想モデルの特性に一致するようにコントローラを適応的に構成しようとするものである．可変パラメータを内蔵した形で適当な構造のコントローラを構成し, プラントの出力と一致するようにコントローラのパラメータが調節される．

しかし, MRACS が代表となる適応制御は適応アルゴリズムが複雑であり, n 次 1 入出力連続系の MRACS においては適応コントローラを構成する積分器は約 $4n$ 個必要となり, それに伴い制御パラメータの数も多くなり, 演算回数が増大するという問題点が指摘され

ている [13]. そこで適応コントロー ラの簡素化を目的としたのが SAC (Simple Adaptive Control - 単純適応制御) である.

SAC は Sobel らによって最初に提案された [1][6][7][8][9]. 「プラント伝達関数に出力フィードバックゲインを施した場合に得られる閉ループ伝達関数が強正実 (SPR) になる」という条件（これを ASPR 条件という）の下で構造の簡単な適応制御系が構成できるというものであり,

- 規範モデルに低次モデルを選択できる.
- ある種のロバスト性を有している. (MRACS では次数のミスマッチがあれば制御系が不安定となることがあるのに対し, SAC では閉ループ系が強正実となる定フィードバックゲイン k_e^* の存在さえ確認できれば制御系の安定性が確保されるので, この意味ではロバストである).
- 適応コントローラの次数をプラント次数に依存しない形式で簡単に構成できる.
- 伝達特性の指定をフィードフォワードループを通じて行なっているので一種の 2 自由度制御構造をとっている.

などの特長がある. これは, 標準の MRACS では見られない優れた性能である. プラントの数式モデルの次数が未知であっても制御可能であり, また, ロバスト性も有している. 不確かさの存在する制御実験等において摩擦等のような外乱の影響の克服に特に有効である.

しかし, この単純適応制御を多入力多出力 (MIMO = Multi-Input Multi-Output) 系[1] に拡張した場合, 一般の制御手法と比較して, 必ずしも優れた結果が得られているとは断言できない. MIMO 系に拡張した場合, システムが大きくなるにつれて, 計算量も大きくなってしまう. 制御系 (コントローラ) の構成が複雑になることも, 依然行われていた実験の結果からも判明した. 特に, コントローラのメモリ使用量や制御信号を決定する計算量が膨大になり, 計算時間も長くなった.

このように, MIMO 系には計算量や計算速度の問題などが発生する可能性が高く, これらの問題を克服する必要があるが, 現在のところ, 多変数単純適応制御の計算量の削減に関する研究は極めて少ない. 入力および出力の数が多くなった場合, そのシステムの構造

[1] 多変数システムとも呼ぶ

によっては, いくつかのブロックに分割することができる. 大規模システムに対して一つの大きて複雑なコントローラを設計するよりも, 各ブロックに対して, 小さなコントローラを構成することで分散的に制御を行った方が, 性能的には解析がしやすく, また, 計算量および計算速度の面から見ればコスト的にも期待できると考えられる.

もう一つの問題は, 標準の MRACS と同じように, 単純適応制御は基本的に線形のプラントのために開発されている. 単純適応制御を ロボット マニピュレータ のように非線形性を含んだシステムに適用しようとした時に, その非線形性を補償できるかどうかというような疑問が湧いてくるであろう. また, 多関節のロボットの制御には, 複数の入力および出力があり, 基本的に多変数システムの構造をとっているので, 軸間に干渉が生じる可能性が高い. よって, これらの項に対する何らかの補償機構が必要となる.

最近, 単変数システムの結合として分散的な適応制御が考案されている [5] が, システムが大変大きくなった場合, 複数のサブシステムブロックで構成された大規模システムについては, 未だに報告されていない. また, 単純適応制御の多関節ロボットの多軸制御への適用についても現在のところまだ公表されていない.

そこで本研究では, 次の点について MIMO-SAC の設計を改善し, シミュレーション的な検討を行なう.

1. 多入出力単純適応制御 (MIMO-SAC) の構成および数値シミュレーションを行い, その結果を分析する.

2. 多入出力単純適応制御 (MIMO-SAC) におけるプラントの構造を考慮し, MIMO-SAC の高速化を図る.

3. MIMO-SAC をロボット マニピュレータの制御に適用する場合の制御系の理論的な設計を行い, その有効性を機構解析プログラム DADS および Matlab によるシミュレーションで検討する.

その結果, 次のような成果が得られている.

1. 低次数 (例えば 3 入力 3 出力) システムの場合, 標準の MIMO-SAC はその威力を発揮しているが, 大規模なシステムにおいては, 計算速度が飛躍的に低下していく.

2. 制御対象の構造を考慮すれば, 標準の MIMO-SAC において余分なコントローラを取り除くことができる. サブシステム間の相互関係より, 全体システムの伝達関数

行列の構造が決定されるので, これに注目し, 簡略化のためのアプローチを行う. これによって, 計算量を軽減することができ, 制御の高速化につながる.

3. 単純適応制御のロボットへの適用について, 非線形性を補うために, 非線形性補償機構の設計法を考案する.

4. 多関節ロボット制御において, 軸間に発生する干渉項は SAC 本来のアルゴリズムによって補償することができるが, この干渉項が有界であれば, 非線形性補償機構と併合した形で補償器が構成できることが分かった. このとき, システムが完全な分散型となり, 計算量の一番少ない形となる.

また, 考案した手法を用いて, さらに超多自由度ロボットの制御への挑戦を試み, DADS によるシミュレーションの結果を報告する.
以下,

- 第 2 章では単純適応制御 (SAC) の一般構成,
- 第 3 章では多入出力単純適応制御 (MIMO-SAC) の構成,
- 第 4 章では プラントの構造を考慮した MIMO-SAC の高速化
- 第 5 章では ロボットマニピュレータのための単純適応制御
- 第 6 章ではロボットのための単純適応制御のシミュレーション結果

を述べる.

付録には本研究で行なったシミュレーションプログラムのソースファイル等を示す.

2 単純適応制御

2.1 適応制御の問題点と SAC の特徴

2.1.1 適応制御手法における問題点

適応制御手法が検討され始めてから既に長年月が経過し, その間の理論的発展には目覚しいものがあった. 特に MRACS, STR [13] を中心とした適応制御理論の基本的な体系化は既に完了していると言っても過言ではない. また, 理論の実際面への応用に関しても活発的な検討が続けられており, 成果の得られている場合も多い.

しかし, それが実システムへの応用と言う点で, 他の制御手法と比較して, 一般論としてより効果的かつ信頼のおける制御技術であるかどうかという点については, 必ずしも肯定的評価のみを与えられない面があることも残念ながら事実である. その理由としてまずあげることができるのは, それが形式としてはフィードバック制御系構造をとっているにもかかわらず, 期待していたほどロバスト性がないという点である. 通常の適応制御系構成で安定性を保証しようとする時,

- プラントの線形性
- 次数既知
- プラントの逆安定性
- パラメータ値の時不変性

などのような条件が必要となるが, 実プラントでは, これらの条件が全て満たされる場合はほとんどなく, その場合, 安定性は保証されず, 制御結果も大きく落ちる. 次の問題点は, 適応コントローラ構成の複雑さである. たとえば連続系に対する適応制御系では, 一般にコントローラ内部にフィルタとパラメータ同定器を含むが, 1 次系のプラントに対する適応コントローラに 7 個の積分器を必要とする方法がある (文献 [13],[14] 参照) ことからわかるように, 多くの積分器が内蔵されなければならないことを意味している. このようなコントローラの複雑さは当然, 制御系の信頼性を低下させる要因となる. また, それは実装時における 1 サンプリング時間内における計算量を, 特に高次プラントに関しては飛躍的に増大させる原因となり, けっして好ましいことではない.

2.1.2 ロバスト適応制御の登場

上述の問題を解決するには, すでに努力が払われており, 特にロバスト適応制御については最近多くの研究がなされてきている [14]. ただ, それが応用技術者の関心をひきつけるものであるかどうかということについては疑問の余地がある.

その理由として,

1. ロバスト適応制御アルゴリズムは, 基本的に従来の適応アルゴリズムに何らかのロバスト制御機構を付加した形式で構成されることである. しかし, ロバスト機構を付加したことにより, システムがますます複雑となる.

2. 多くのロバスト適応アルゴリズムが, 低次元化に伴う寄生要素対策, 外乱, パラメータの時変性, といった阻害要因に個別に対応する形式となっているが, 実プラントではそれらは総合された形で存在することが多い.

2.1.3 単純適応制御

上述の問題のうち, 特に適応コントローラの簡素化に関しては, 単純適応制御 (SAC - Simple Adaptive Control, または Simplified Adaptive Control とも呼ばれる) が有効である.

単純適応制御は Sobel, Kaufman および Mabius [1] により最初に提案されたもので, プラントが ASPR (Almost Strictly Positive Real), つまり, プラント伝達関数に定ゲイン出力フィードバックを施した場合に得られる閉ループ伝達関数が強正実 (SPR) になる,

という条件下で簡単な適応制御系が構成できる, というものである. その後, この考えは Bar-Kana および Kaufman [6][7][8][9] によって拡張された.

SAC は通常のモデル規範型適応制御 MRACS に見られない特徴をもっている. この方法では,

- 規範モデルを低次モデルに選ぶことが可能
- 適応コントローラの次数をプラント次数に依存しない

形式で制御系が簡単に構成できる. また,

- 通常の MRACS では次数のミスマッチが生じれば制御系が不安定となる.
- SAC では閉ループ系が強制実となるようなゲイン k_e^* を実際に求めなくても, 存在さえ確認できれば制御系の安定性が保証される.

の意味では SAC 自体はある種のロバスト性を有している. これは, 一般の直接法による MRACS ではプラント次数情報と逆安定性の仮定を用いて, 誤差システムが強正実となる制御系を構成したのに対し, 単純適応制御 SAC ではプラントが概強正実 (ASPR) であればある出力ゲイン k_e^* の出力フィードバックのみで誤差システムを強正実にできることに注目し, その未知ゲインを推定しているという点が (従来の MRACS と) 異なる.
さらに, SAC では

- 制御系の伝達特性の指定をフィードフォワードループを通じて行っている

ので, これは SAC が一種の 2 自由度制御系構造をとっていることを意味している.

基本の単純適応制御 (SAC) は ASPR なプラントに対してのみ適用可能であるが, 非 ASPR プラントに対しても, 並列フィードフォワード補償 (PFC) を施すことで, 制御対象を ASPR 系で近似できる [6][7][8][9].

制御系設計に用いる制御対象のモデルと実制御対象とが, 厳密に一致していなくても, 使用する制御アルゴリズムが何らかの意味でロバスト性をもっていれば, 実用上十分な範囲で制御系が構成できる. SAC は適応制御アルゴリズムのうちでも比較的ロバスト性に富んだアルゴリズムであるといえる. この事実は Bar-Kana によって行われた多くのシミュレーション例からも推察できる.

この SAC に, 外乱などのような, 非線形性を含んだプラントの要素を補償するために, ロバスト適応制御機構を付加することで, さらにロバスト化でき, 制御性能を向上ささることができる.

2.2 SAC の基本構成

SAC の構成図は図 2.1 に示す. 本研究では主に多入出力系の制御について扱っているが, 単純適応制御の基本構成を説明するために, まず, 一入出力単純適応制御系 (SISO-SAC = Single-Input Single-Output Simple Adaptive Control) の構成を述べることにする [2]. なお, 多入出力系のための単純適応制御については第 3 にて説明する.

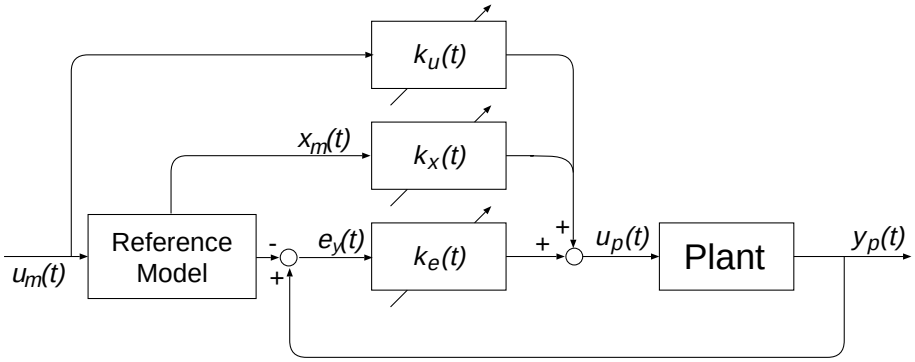


図 2.1: SAC の基本構成

いま, プラントが可制御可観測な n_p 次 1 入出力線形系とする.

$$\dot{x}_p(t) = A_p x_p(t) + b_p u_p(t) \tag{2.1}$$

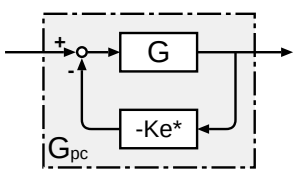
$$y_p(t) = {c_p}^T x_p(t) \tag{2.2}$$

A_p, b_p, c_p は未知パラメータをもつ行列とベクトルである. プラントは次の仮定を満足しているとする.

仮定 2.1

1. プラントは ASPR である. つまり, ある定数 k_e^* が存在し, $A_{pc} = A_p + b_p k_e^* {c_p}^T$ とするとき,

$$G_{pc}(s) = {c_p}^T (sI - A_{pc})^{-1} b_p \tag{2.3}$$



は SPR (強正実) である.

2. つぎのシステム行列

$$M = \begin{bmatrix} A_p & b_p \\ {c_p}^T & 0 \end{bmatrix} \tag{2.4}$$

は正則である.

以上の仮定の下で, 追従すべき n_m 次 $(n_m \leq n_p)$ の漸近安定な規範モデルを

$$\dot{x}_m(t) = A_m x_m(t) + b_m u_m(t) \tag{2.5}$$

$$y_m(t) = {c_m}^T x_m(t) \tag{2.6}$$

と与え, 制御目的は

$$e_y(t) = y_p(t) - y_m(t) \tag{2.7}$$

$$\lim_{t\to\infty} e_y(t) = 0 \tag{2.8}$$

を実現することである. **操作入力** は

$$u_p(t) = k_e(t)e_y(t) + k_x(t)^T x_m(t) + k_u(t)u_m \tag{2.9}$$

と与え, **適応同定則** として外乱に敏感な微分型手法を避け, 比例積分則を用いる.

$$\begin{cases} K(t) = K_I(t) + K_P(t) \\ \dot{K}_I(t) = -\Gamma_{\theta I}^{-1} z(t) e_y(t) - \sigma_\theta(t) K_I(t) \\ K_P(t) = -\Gamma_{\theta P}^{-1} z(t) e_y(t) \end{cases} \tag{2.10}$$

$$\begin{cases} z(t) = [e_y(t), x_m(t)^T, u_m(t)]^T \\ K(t) = [k_e(t), k_x(t)^T, k_u(t)]^T \\ \sigma_\theta(t) = \frac{\sigma_\theta e_y^2(t)}{1+e_y^2(t)} \quad \sigma_\theta > 0 \\ \Gamma_{\theta I}, \Gamma_{\theta P} : \text{正定対称なゲイン行列} \end{cases} \tag{2.11}$$

概念的には, $K(t)$ では $k_e(t)$, $k_x(t)$, および $k_u(t)$ の値を適応的に推定し, $k_e(t)$ は k_e^* に近づけるように調整していく. マッチング状態が達成されたら, プラントのへいループ系が SPR となる. また, プラント伝達特性の指定は $k_x(t)$, および $k_u(t)$ を用いてフィードフォワードループを通じて行なっている. これが, SAC が 2 自由度制御構造をとっていることを表している.

2.3 フィードフォワード補償

プラントが非 ASPR の場合には, プラント伝達関数にフィードフォワード補償 (FC[1]) d_p を施し,

$$G_a(s) = G_p(s) + d_p \tag{2.12}$$

が ASPR となるようにする. このとき, d_p を十分小さく取れば, フィードフォワード補償による影響は実際上ほとんどない. 一入出力の場合はとりあえずこれで良いが, 多入出力系に拡張した場合は, 並列にフィードフォワード補償を構成することで, 組織的にフィードフォワード補償[2] を施す必要がでてくる. これは第 3 章で述べることにする.

2.4 ロバスト単純適応制御 (Robust SAC)

SAC は従来の PID 制御よりは比較的ロバスト性に優れている. これに簡単なロバスト適応制御機構を付け加えることで, さらにロバスト化し, かつ制御機能を向上させることが出来る. すなわち, 次のように操作入力にロバスト機構 $u_r(t)$ を付け加えることによりロバスト SAC を構成する [2][7].

$$u_p(t) = K(t)^T z(t) + u_r(t) \tag{2.13}$$

$$u_r(t) = \begin{cases} -\rho(t)\text{sgn } e_y(t), & \rho(t)|e_y(t)| \geq \epsilon \\ -\rho(t)^2 e_y(t)/\epsilon, & \rho(t)|e_y(t)| < \epsilon \end{cases} \tag{2.14}$$

$$\begin{cases} \rho(t) &= \rho_P(t) + \rho_I(t) \\ \rho_P(t) &= k_P|e_y(t)| \\ \dot{\rho}_I(t) &= -\sigma_I(t)\rho_I(t) + k_I|e_y(t)| \end{cases} \tag{2.15}$$

$$\begin{aligned} \epsilon &: \text{chattering 緩和微小正数} \\ k_I, k_P(>0) &: \text{制御ゲイン} \\ \sigma_I(t) &: \sigma_\theta(t) \text{ と同様に構成} \end{aligned} \tag{2.16}$$

[1] Feedforward Compensator

[2] 並列フィードフォワード補償 (PFC = Parallel Feedforward Compensator) と呼ぶ

3 多入出力単純適応制御系 (MIMO-SAC) の構成

—並列フィードフォワード補償に基づく一般設計—

3.1 多入出力 SAC の基本構成

SAC は、ASPR なプラント、すなわち、定ゲイン出力フィードバックにより強正実 (SPR) 化可能なプラントに対してのみ適用可能である. 非 ASPR プラントに並列フィードフォワード補償 (PFC = Parallel Feedforward Compensator) を施すことで、制御対象を ASPR 系で近似できる可能性がある [6][7][8][9].

岩井 [2][3] らの提案によれば、1 入出力系に対するフィードフォワード補償の構成法の拡張の形で、多入出力系 PFC が構成できる. その拡張は自明ではないが, 次の条件の下でこの設計法は、

1. プラント伝達関数の相対次数の上限値 γ^* が既知
2. プラントは最小位相[1]
3. プラント伝達関数の最高位係数 (leading coefficient) 及び低周波ゲインの概略値が既知

であれば、$(\gamma^* - 1)$ 個のフィードフォワード補償を多重並列に構成することで、制御対象を ASPR 系で近似できるものである.

[1] ASPR の場合は当然最小位相系である

3.1.1 制御対象・規範モデル

SAC の基本構成は 2 章で述べた（概略図 2.1 参照）. 次は、多変数単純適応制御系 (MIMO-SAC = Multi-Input Multi-Output SAC) の構成について記述する. プラントが可制御可観測な n_p 次 m 入出力線形系 $(n_p \geq m)$ とする.

$$\dot{x}_p(t) = A_p x_p(t) + B_p u_p(t) \tag{3.1}$$

$$y_p(t) = C_p x_p(t) \tag{3.2}$$

A_p, B_p, C_p は未知パラメータをもつ行列であり、$x \in R^n, u, y \in R^m$ は、それぞれ、状態及び入出力ベクトルである. 追従すべき n_m 次 $(n_m \leq n_p)$ の m 入出力漸近安定な規範モデルを

$$\dot{x}_m(t) = A_m x_m(t) + B_m u_m(t) \tag{3.3}$$

$$y_m(t) = C_m x_m(t) \tag{3.4}$$

と与え, 制御目的は

$$e_y(t) = y_p(t) - y_m(t) \tag{3.5}$$

$$\lim_{t\to\infty} e_y(t) = 0 \tag{3.6}$$

を実現することである.

仮定 3.1

プラント及び規範モデルは次の仮定を満足しているものとする.

1. プラントは $(g(t) \equiv 0$ のとき) ASPR である. つまり, ある定ゲイン行列 K_e^* が存在し, $A_{pc} = A_p + B_p K_e^* C_p$ とするとき,

   $$G_s(s) = C_p(sI - A_{pc})^{-1} B_p \tag{3.7}$$

   は SPR (強正実) となる.

2. プラント及び規範モデルは、Broussard [10] のモデル出力追従条件を満たす.

   $$\begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{bmatrix} = I \tag{3.8}$$

   なる Ω_{ij} が存在し、Ω_1 の固有値は A_m の固有値の逆数と一致しない.

3. $u_m(t)$ は、その微分値 $\dot{u}_m(t)$ を入力とする線形定数係数安定系の出力が有界となるような規範入力.

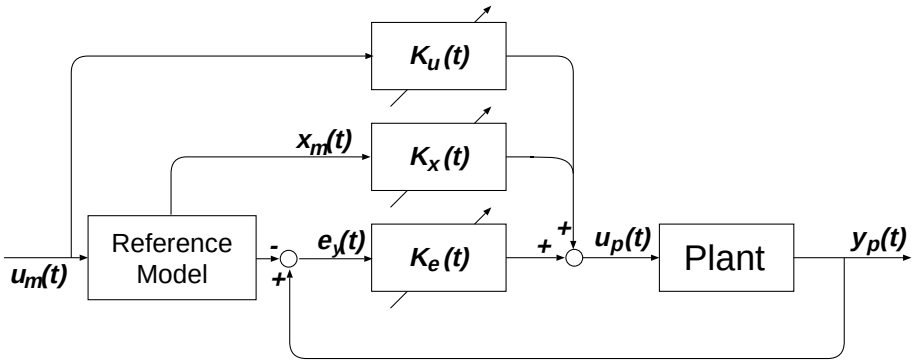


図 3.1: 多入出力 SAC の基本構成

3.1.2 制御入力・適応同定則

制御目的を達成するため、この仮定のもとで、制御入力を以下のように、適応的に構成する. これもまた、適応同定則として比例積分則を利用する.

$$u_p(t) = K(t)z(t) \tag{3.9}$$
$$z(t) = [e_y(t)^T, x_m(t)^T, u_m(t)^T]^T \tag{3.10}$$
$$K(t) = [K_e(t), K_x(t), K_u(t)] \tag{3.11}$$
$$K(t) = K_I(t) + K_P(t) \tag{3.12}$$
$$\dot{K}_I(t) = -e_y(t)z(t)^T\Gamma_I - \sigma_I(t)K_I(t) \tag{3.13}$$
$$K_P(t) = -e_y(t)z(t)^T\Gamma_P \tag{3.14}$$
$$\sigma_I(t) = \sigma_1\frac{e_y(t)^Te_y(t)}{1+e_y(t)^Te_y(t)} + \sigma_2, \quad \sigma_1,\sigma_2 > 0 \tag{3.15}$$
$$\Gamma_I,\Gamma_P \ : \ \text{正定対称なゲイン行列} \tag{3.16}$$

3.2 並列フィードフォワード補償によるプラントの ASPR 化

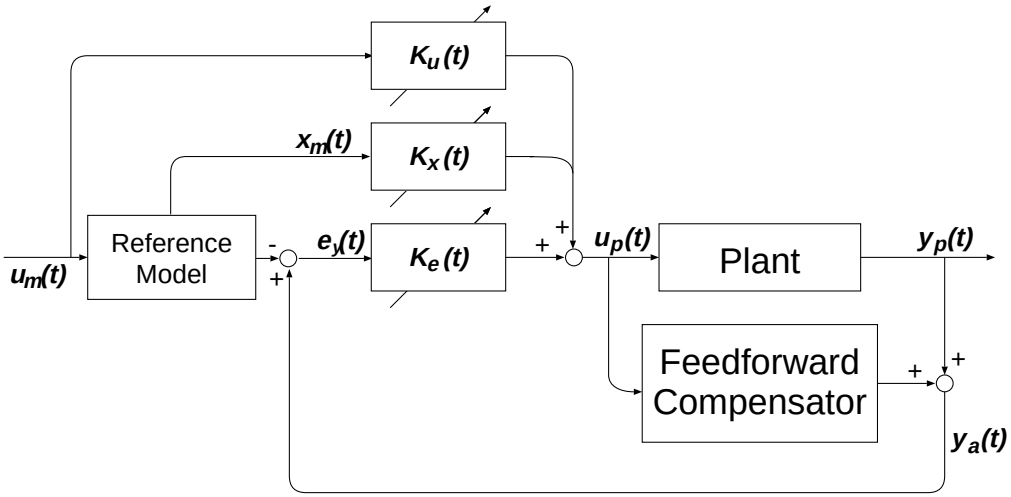


図 3.2: PFC を施した SAC の構成

以上、プラントが ASPR である場合の SAC の基本構成について述べたが、実際の多くのプラントは ASPR 条件を満足しない非 ASPR プラントである. この場合、PFC を施すことで、SAC が適用可能となる.

プラントが非 ASPR の場合、図 3.2 に示すように、プラント伝達関数にフィードフォワード補償 $F(s)$ を施し,

$$G_a(s) = [g_{a_{ij}}(s)]_{i,j=1,\ldots,m} = G_p(s) + F(s) \tag{3.17}$$

$$G(s) = G_p(s) = [g_{ij}(s)]_{i,j=1,\ldots,m} = C(sI - A)^{-1}B = \frac{1}{p(s)}\Phi(s) \tag{3.18}$$

を考え、条件

1. $G_a(s)$ は ASPR

2. ある $\epsilon > 0$ を与えた時、$0 \leq \omega \leq \omega_0$ なる周波数帯域で

$$\sum_{i=1}^{m}\sum_{k=1}^{m} |\{|g_{a_{ik}}(j\omega)|\} - \{|g_{ik}(j\omega)|\}| \leq \epsilon \tag{3.19}$$

を満足するように $F(s)$ を設計する. 対象プラントは、以下の仮定を満足しているものとする.

仮定 3.2

(1) 最小位相系である. すなわち、プラント零点多項式 $z(s)$ は Hurwitz(monic) 多項式.

(2) $\Phi(s)$ の任意の r 次主小行列式 (r=1,...,m) の最高位係数は正.

(3) G(s) の (i,j) 要素 $g_{ij}(s)$ の相対次数 $\gamma_{ij}(\gamma_{ij} \geq 2, i \neq j)$ は既知、また、$g_{ij}(s)$ の最高位係数の概略値は既知.

(4) G(s) の極及び零点多項式の次数と対角要素の相対次数 $\gamma_i(=\gamma_{ii})$ との間に次の関係が成立する.

$$\deg p(s) - \deg z(s) = d \leq \sum_{i=1}^{m} \gamma_i = d_0 \tag{3.20}$$

(5) $|g_{ii}(j0)|, (i = 1, ..., m)$ の概略値は既知.

補題 3.1

PFC の設計 :

仮定 2(1),(2),(4) を満足するプラント (3.17) に対し、フィードフォワード補償

$$F(s) = \frac{1}{p_f(s)} \mathrm{diag}[\rho_1 f_1(s), ..., \rho_m f_m(s)] \tag{3.21}$$

$$p_f(s) : n_f\text{次 Hurwitz 多項式 } (F(s) \text{ の極多項式}) \tag{3.22}$$

$$f_i(s) : (n_f - \gamma_i + 1)\text{ 次 monic 多項式} \tag{3.23}$$

を施す. ただし、$\rho_i, i \in M = 1, ..., m$ は、正、かつ、多項式

$$\sum_{1 \leq i_1 < ... < i_h \leq m} \left\{ \left(\prod_{\substack{j=1 \\ j \neq i_1 ... i_h}}^{m} \rho_j f_j(s) \right) \Phi[i_1 ... i_h] \right\} \tag{3.24}$$

の最高位係数を $b_h(h = 1, ..., m)$ とする時、

$$\tilde{b} = b_m \gg b_{m-1} \gg ... \gg b_1 \gg b_0 = \prod_{i-1}^{m} \rho_i \tag{3.25}$$

となるように与える. $\Phi[i_1 ... i_h]$ は、$\Phi(s)$ から第 $i_1 ... i_h$ 行及び列を残し、それ以外の行及び列を除き構成される $\Phi(s)$ の h 次主小行列式である. この時、$G_a(s)$ の対角要素の相対次数は 1 減少する.

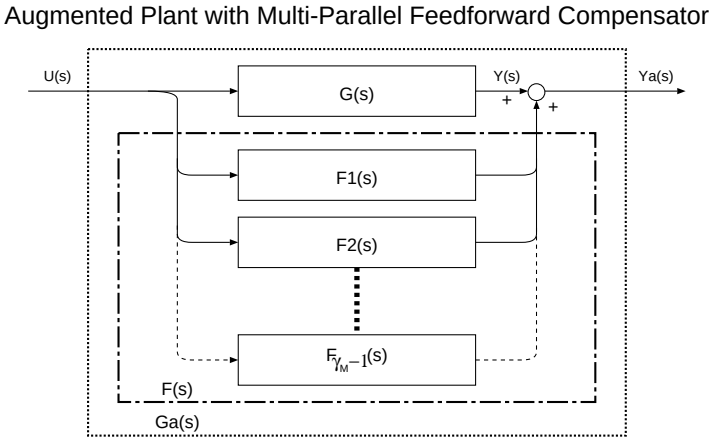


図 3.3: PFC の構成

補題 3.2

前述の補題において、$\rho_i > 0, i \in L, \rho_i = 0, i \in N$ $(L + N = M)$ とする. この時、$G_a(s)$ は仮定 3.2 (1),(2),(4) を満足し、対角 $(i,i), i \in L$ 要素の相対次数が 1 減少する.

定理 3.1

仮定 3.2 を満足する非 ASPR プラントを考える. このプラントに対し、並列フィードフォワード補償 F(s) を次のように構成する.

$$F(s) = \sum_{i=1}^{\gamma_M - 1} F_i(s), \gamma_M = \max_{i \in M}(\gamma_i) \tag{3.26}$$

$$F_i(s) = \mathrm{diag}[\rho_{i1}/d_{i1}(s), ..., \rho_{im}/d_{im}(s)] \tag{3.27}$$

$$d_{ij}(s) : (\gamma_j - i) \text{ 次 monic 安定多項式} \tag{3.28}$$

$$\rho_{ij} : \begin{cases} \rho_{ij} > 0 & (\gamma_j - i) > 0 \\ \rho_{ij} = 0 & (\gamma_j - i) \leq 0 \end{cases} \tag{3.29}$$

かつ、

$$G_{ai}(s) = G_{ai-1}(s) + F_i(s), i = 1, ..., \gamma_M - 1 \tag{3.30}$$

$$G_{a0}(s) = G(s) \tag{3.31}$$

とおく. さらに、プラント対角要素の低周波ゲインとの間に、

$$|g_{kk}(j0)| \gg \sum_{i=1}^{\gamma_M - 1} |\rho_{ik}/d_{ik}(j0)|, k \in M \tag{3.32}$$

なる関係が成立するように選ぶ. この時、拡張系

$$G_a(s) = G_{a\gamma_M - 1}(s) = G(s) + F(s) \tag{3.33}$$

は、条件 1、条件 2 をすべて満足する. 但し、G(s)+F(s) で極零消去は起こらないものとする.

以上, 多入出力単純適応制御について述べたが, これを実際の多入出力のプラントに適用すれば, 「膨大な計算量」という問題が現れてくる. これを克服するための方法を次章で述べる.

4 プラントの構造を考慮した MIMO-SAC の高速化

— 多入出力単純適応制御における計算量の軽減 —

4.1 はじめに

適応制御は適応アルゴリズムが複雑であり, n 次 1 入出力連続系の MRACS においては適応コントローラを構成する積分器は約 $4n$ 個必要となり, それに伴い制御パラメータの数も多くなり, 演算回数が増大するという問題点も指摘されている [13][14]. そこで適応コントロー ラの簡素化を目的としたのが SAC (Simple Adaptive Control - 単純適応制御) である.

多入力多出力単純適応制御 (MIMO-SAC = Multi Input Multi Output Simple Adaptive Control) による大規模システム (LSS = Large Scale System) 的な制御シミュレーションを行った. また, 2 変数 (2 入力 2 出力) および 3 変数 (3 入力 3 出力) システムに対する制御実験も良好な結果を与えた. その結果をさらに解析し, サブ・システム間の干渉が補償されることが確認できたが, システムが大きくなる[1]につれて, 制御系 (コントローラ) の構成が複雑になることも, 実験結果から判明した. 特に, コントローラのメモリ使用量や制御信号を決定する計算量が膨大になり, 計算時間も長くなった. システムが大きくなると, 制御信号を計算するのに時間がかかり, 次周のサンプリング時刻に間に合わない恐れがでてくる.

[1] 入出力の数が大きくなる

大規模システムにおいて, 大変複雑なコントローラの構成を簡略化する必要がでてくる. しかし, 残念ながら今までのところ多変数適応制御の簡略化に関する研究はほとんど行われていない. 堂園 [5] らが分散的な適応制御の特殊な一構成を提案しているが, 複数のサブシステムブロックで構成された大規模システムについては適用可能かどうかまだ検討されていない.

そこで簡略化のために, 別の観点からアプローチを行う. サブシステム間の相互関係より, 全体システムの伝達関数行列の構造が決定されるので, このことに注目し, システムの簡略化を図る. この章では, この方法でシステムの簡略化を実現できることを示す.

4.2 大規模システムにおける多変数適応制御コントローラの計算量

多入出力単純適応制御は, 標準の MRACS (モデル規範型適応制御) より比較的計算量が少ないが, それでも, 制御対象のスケールが大きくなると, やはり計算量の数は爆発的に大きくなるのが悩みの種である.

この MIMO-SAC アルゴリズムの各々の計算量を表 4.1 に示す.

表 4.1: MIMO-SAC の計算量

個数	掛算器	加算器	積分器
規範モデル	$\mathrm{O}(n_m(n_m+m))$	$\mathrm{O}(n_m(n_m+m))$	n_m
適応同定	$\mathrm{O}(m(n_m+2m)^2)$	$\mathrm{O}(m(n_m+2m)^2)$	$m(2m+n_m)$
PFC	$\mathrm{O}(m)$	$\mathrm{O}(m)$	$\mathrm{O}(\gamma m)$
ロバスト機構	$\mathrm{O}(m^2)$	$\mathrm{O}(m)$	m

これは, 一回の (制御入力を決定する) ループで必要な演算器を表す. 表から分かるように, システムが大きくなる, すなわち入出力の数が大きくなると掛け算器の数は 2 乗というオーダで増加する. これを, 1 乗のオーダに抑えることができれば, 計算時間も短縮できることを言うまでもないのであろう.

ところで, ある程度制御対象の構造が分かっていれば, その構造を考慮し, より簡潔なコントローラを設計することが考えられる. その可能性が実現可能であることを, 以下に

示す.

4.3 フィードフォワード型モデル追従制御問題

4.3.1 フィードフォワード型コントローラ

モデル追従型制御問題を解くには様々な方法があるが, その中の一つ, フィードフォワード型コントローラに基づく方法を述べる.

連続・線形・時不変系プラント

$$\begin{aligned} \dot{x} &= A_p x_p + B_p u_p \\ y &= C_p x_p + D_p u_p \\ x &\in R^{n_p}, x \in R^{m_p}, x \in R^l \end{aligned} \tag{4.1}$$

が与えられたとする. 但し, A_p, B_p, C_p, D_p は適切な大きさをもつ行列であり, (A_p, B_p) は可制御で, (A_p, C_p) は可観測であると仮定している.

これに対して, 追従すべき規範モデルを

$$\begin{aligned} \dot{x} &= A_m x_m + B_m u_m \\ y &= C_m x_m + D_m u_m \\ x &\in R^{n_m}, x \in R^{m_m}, x \in R^l \end{aligned} \tag{4.2}$$

と設定する.

仮定 4.1

1. プラントは可制御, 可観測, SPR である
2. 規範モデルは, 漸近安定である
3. 規範モデルの入力は定数または漸近安定なモデルの出力で表せるものである
4. 追従可能なプラントは, 完全追従が達成された時の状態および入力は規範モデルの状態および入力項の線形結合で表せる

このとき, 次のことが成り立つ.

定理 4.1

プラントの伝達関数行列 $(l \times m_p)$ が

$$G_p(s) = C_p(sI - A_p)^{-1}B_p + D_p \tag{4.3}$$

また, 規範モデルの伝達関数行列 $(l \times m_m)$ が

$$G_m(s) = C_m(sI - A_m)^{-1}B_m + D_m \tag{4.4}$$

と表現できるとき,

$$G_p(s)G_c(s) = G_m(s) \tag{4.5}$$

なるフィードフォワード補償型コントローラ $G_c(s), \quad (m_p \times m_m)$ がが存在する.

証明

(3.8) 式および Broussard モデル [10] より, 完全追従が達成されたとき, そのときの状態および入力を x^* および u^* と表すと,

$$\begin{bmatrix} x^* \\ u^* \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1,k+2} & \cdots \\ S_{21} & S_{22} & \cdots & S_{2,k+2} & \cdots \end{bmatrix} \begin{bmatrix} x_m \\ u_m \\ \dot{u}_m \\ \vdots \\ u_m^{(k)} \\ \vdots \end{bmatrix} \tag{4.6}$$

が成立し, (3.8) 式の解が

$$G_c(s) = S_{21}[sI - A_m]^{-1}B_m + s\Omega_{21}[I - s\Omega_{11}]^{-1}S_{12} + S_{22} \tag{4.7}$$

である. ∎

この定理は, 後述する SAC の簡略化方法に役に立つ. また, 次のことが言えることも言うまでもないのであろう.

補題 4.1

漸近安定な規範モデルをおき, プラントが SPR の意味で安定であるとき, フィードフォワードコントローラで補償されたシステム (4.5) が安定となる.

4.3.2 SAC の簡略化：その 1

G_p $(l \times m_p)$ をプラントの伝達関数, G_m $(l \times m_m)$ を規範モデルの伝達関数, G_c $(m_p \times m_m)$ をモデル追従補償器の伝達関数とすし, 完全追従が達成されたとき, (4.5) の関係が成り立つ.

モデル追従型適応制御系における問題はこの G_c を推定することである. ただし, 一般に $l \leq m_p, l \leq m_m$ であるが, ここで $l = m_p = m_m$ とおくことにする. そうすれば, G_p が正則であれば,

$$G_c = G_p^{-1} G_m \tag{4.8}$$

となる.

SAC の計算量削減策の準備として, 上記の定理を用い, 次の補題を導く.

補題 4.2

G_m の構造を最も単純な形 (最小型非干渉系) に選び, G_c をプラントの構造[2] にしたがって構成する. 具体的に, $n_m = m$ で,

$$G_m(s) = \mathrm{diag}[\frac{b_i}{s + a_i}]_{i=1}^{m} \tag{4.9}$$

である. G_m が対角行列であることから,

$$G_{cij}(s) = 0, \quad (i \neq j, G_{p_{ij}}^{-1} \approx 0) \tag{4.10}$$

とできる.

証明

後述する具体例でも証明するが, G_m は既知であるから, G_p^{-1} の構造さえ分かれば, G_c を構成することができる. また, 例題からでも分かるように, 大規模システムにおいて, 各サブシステム間のつながり具合が把握できれば, ${G_p}^{-1}$ の構造が抽出できる. 例えば, ${G_p}^{-1}$ が帯状の形式を取っていれば, G_m も帯状の形になる. ∎

[2] ここで言う「プラントの構造」とは, プラントの状態方程式より得られた伝達関数行列の形式をさしている.

4.4 単純適応制御：多自由度型制御

4.4.1 SAC の特徴

SAC は, プラントの出力のフィードバックを使って制御信号を決めるだけでなく, 規範モデルの入力および状態パラメータが加わり, 制御信号として多自由度の制御系を構成する.

SAC の特徴より,

- MRACS では次数のミスマッチがあれば制御系が不安定となることがあるのに対し, SAC では閉ループ系が強正実となる K_e^* の存在さえ確認できれば制御系の安定性が確保されるので, この意味ではロバストである.
- 適応コントローラの次数をプラント次数に依存しない形式で構成できる.
- 伝達特性の指定をフィードフォワードループを通じて行なっているので一種の 2 自由度制御構造をとっている.

等がある.

これは, 一般の直接法による MRACS ではプラント次数情報と逆安定性の仮定を用いて, 誤差システムが強正実となる制御系を構成したのに対し, 単純適応制御 SAC ではプラントが概強正実 (ASPR) であればある出力ゲイン $K_e{}^*$ の出力フィードバックのみで誤差システムを強正実にできることに注目し, その未知ゲインを推定しているという点が (従来の MRACS と) 異なる.

さて, 上記の定理は SPR なプラントに対して有効であるが, ASPR なプラントに対しても有効であるかどうか検討する必要がある.

定理 4.2

ASPR なプラントに対して, 出力フィードバックゲイン $K_e{}^*$ を施したとする. この時, 閉ループシステム ($A_{pc} = A_p + B_p K_e^* C_p$) において, その伝達関数行列の逆行列 G_{pc}^{-1} は, 元のプラントの伝達関数行列の逆行列 G_p^{-1} と同じ構造をとっており, (4.8) 式のように, G_p^{-1} と同様な扱いをすることができ, コントローラ G_c を構成できる.

証明

(2.3) 式と同様に,

$$\begin{aligned} G_{pc}(s) &= (I - G_p(s)K_e^*)^{-1}G_p(s) \\ G_{pc}(s)^{-1} &= G_p(s)^{-1}(I - G_p(s)K_e^*) \\ &= G_p(s)^{-1} - K_e^* \end{aligned} \tag{4.11}$$

K_e^* を G_p^{-1} と同じ形式に選べば G_{pc}^{-1} も同じ構造になる.

$G_{p_{ij}}^{-1} \approx 0, (i \neq j)$ のとき,

$$\begin{cases} K_{e_{ij}}^* = 0 \\ G_{c_{ij}} = 0 \end{cases} \tag{4.12}$$

とできる.

仮定 4.2

仮定 4.1 において SPR 条件を ASPR に置き換える.

4.4.2 SAC の簡略化：その 2

表 4.1 から分かるように, 計算のほとんどは適応同定アルゴリズムが占めている. 以下, SAC アルゴリズムの特徴をいかし, 計算量の削減策について, 具体的な設計法を提案する.

補題 4.3

1. 規範モデルを最小実現型非干渉安定系とする.
2. 適応同定ゲイン行列 Γ を対角行列と選ぶことにする. $\Gamma = \mathrm{diag}[\Gamma_i]_{i=1}^m$.
3. $K_e(t)$ は対称行列であることから, 下三角行列 $K_{eij}, (i = 2, \ldots, m; j = 1, \ldots, i-1)$ を計算する必要がなくなり, $K_{eij} = K_{eji}$.
4. 適応同定ゲイン行列の計算に次のルールを適用する.
   $G_{p_{ij}}^{-1} \approx 0, (i \neq j)$ のとき,

$$\begin{cases} K_{eij}(t) = 0 \\ K_{xij}(t) = 0 \\ K_{uij}(t) = 0 \end{cases} \tag{4.13}$$

証明

補題 4.3 (1 〜 3) は, SAC の構成および次のことより明らかである.

- 1 は, SAC 構成の性質を利用して規範モデルをもっとも簡単な形にする
- 2 では Γ が正定であれば良い
- 3 は適応ゲインの計算より明らか

補題 4.3 の 4 については, 具体的な例などをあげて説明する. 大システムにおける入力信号数と出力信号数は同じと仮定し, これを「変数の数 $(= m)$」と呼ぶことにする. $m = 1$ の場合, システムは SISO 系となり, 各信号の流れが明らかである. $m > 1$ すなわちシステムが多入力多出力系 (MIMO) の場合, 次のように分類し,

- 分離近似型システム
- 直列・連鎖的なシステム
- 多重接続を含んだ一般のシステム

のそれぞれの場合の簡略化の具体案を述べる.

4.5 分離近似型システム

大システムが複数のサブシステムによって構成されているが, そのサブシステム間に干渉が少ない場合, 大規模システムに対して大きなコントローラを構成するよりも, 各サブシステムに対してコントローラを設計し, それらを並列に実行させたほうが良いであろう. このことを簡潔に説明すると以下のようになる.

4.5.1 規範モデル

規範モデルは, やはり最小型非干渉系にする.

$$\dot{x}_{m_1} = a_{m_1} x_{m_1} + b_{m_1} u_{m_1} \quad , \quad y_{m_1} = c_{m_1} x_{m_1}$$

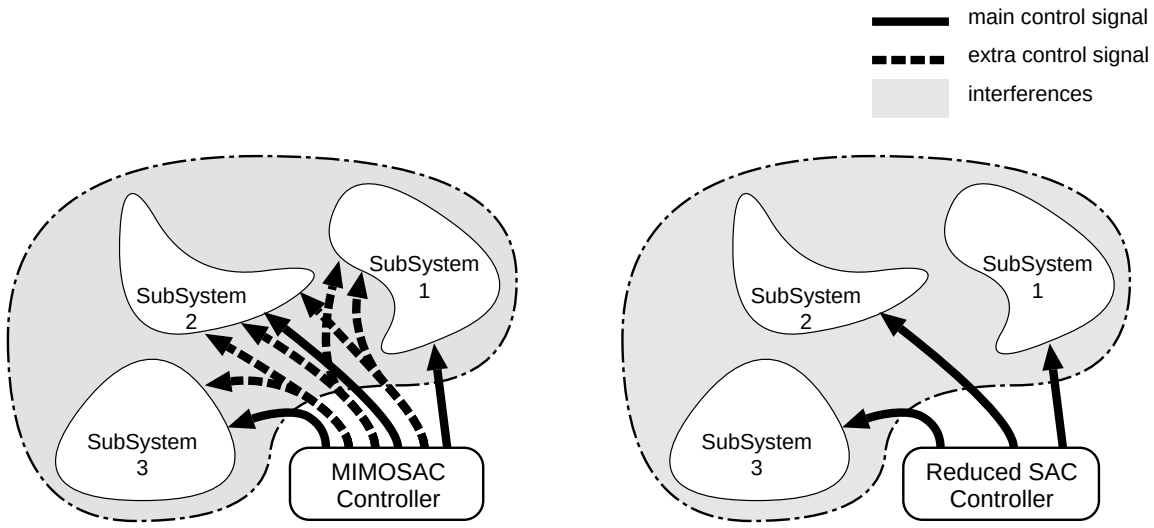


図 4.1: 分離近似型システム

$$\begin{aligned} \dot{x}_{m_2} &= a_{m_2} x_{m_2} + b_{m_2} u_{m_2} \quad , \quad y_{m_2} = c_{m_2} x_{m_2} \\ &\vdots \\ \dot{x}_{m_m} &= a_{m_m} x_{m_m} + b_{m_m} u_{m_m} \quad , \quad y_{m_m} = c_{m_m} x_{m_m} \end{aligned} \tag{4.14}$$

すなわち

$$\begin{aligned} \dot{x}_m = \begin{bmatrix} \dot{x}_{m_1} \\ \dot{x}_{m_2} \\ \vdots \\ \dot{x}_{m_m} \end{bmatrix} &= \begin{bmatrix} a_{m_1} & 0 & \cdots & 0 \\ 0 & a_{m_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{m_m} \end{bmatrix} \begin{bmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_m} \end{bmatrix} \\ &+ \begin{bmatrix} b_{m_1} & 0 & \cdots & 0 \\ 0 & b_{m_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & b_{m_m} \end{bmatrix} \begin{bmatrix} u_{m_1} \\ u_{m_2} \\ \vdots \\ u_{m_m} \end{bmatrix} \end{aligned} \tag{4.15}$$

$$y_m = \begin{bmatrix} y_{m_1} \\ y_{m_2} \\ \vdots \\ y_{m_m} \end{bmatrix} = \begin{bmatrix} c_{m_1} & 0 & \cdots & 0 \\ 0 & c_{m_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & c_{m_m} \end{bmatrix} \begin{bmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_m} \end{bmatrix} \tag{4.16}$$

ただし，

$$n_m = m$$

$$x_m, u_m, y_m \in R^{m\times 1}$$
$$A_m, B_m, C_m \in R^{m\times m}$$

である.

4.5.2 制御対象の記述

プラント (制御対象) は一般に

$$\begin{array}{ccccccccccc} \dot{x}_1 = A_1x_1 & + & b_1u_1 & + & k_{11}y_1 & + & \cdots & + & k_{1m}y_m & ; & y_1 = c_1^Tx_1 \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots & & \\ \dot{x}_m = A_mx_m & + & b_mu_m & + & k_{1m}y_1 & + & \cdots & + & k_{mm}y_m & ; & y_m = c_m^Tx_m \end{array} \tag{4.17}$$

と記述できる. 但し, $k_{ij}y_j$ は干渉項を表す. これを周波数ドメイン表現で表すと

$$\begin{array}{ccccccccccc} (sI - A_1)X_1 = & & b_1U_1 & + & k_{11}Y_1 & + & \cdots & + & k_{1m}Y_m & ; & Y_1 = c_1^TX_1 \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots & & \\ (sI - A_m)X_m = & & b_mU_m & + & k_{1m}Y_1 & + & \cdots & + & k_{mm}Y_m & ; & Y_m = c_m^TX_m \end{array} \tag{4.18}$$

となる.

仮定 4.3

干渉項 $k_{ij}y_j$ は微小である. 具体的に

$$\sum_{j=1}^{m} \|k_{ij}\| \ll \lambda_{\min}[A_i]\|b_i\|$$

この時, 干渉項による影響は入力項に比べて極めて小さいとみなせる.

以上の仮定が成立すれば, 制御対象の状態方程式は

$$\begin{array}{rcl} \dot{x}_{p_1} = A_{p_1}x_{p_1} + b_{p_1}u_{p_1} & , & y_{p_1} = c_{p_1}^Tx_{p_1} \\ \dot{x}_{p_2} = A_{p_2}x_{p_2} + b_{p_2}u_{p_2} & , & y_{p_2} = c_{p_2}^Tx_{p_2} \\ & \vdots & \\ \dot{x}_{p_m} = A_{p_m}x_{p_m} + b_{p_m}u_{p_m} & , & y_{p_m} = c_{p_m}^Tx_{p_m} \end{array} \tag{4.19}$$

と近似できる.

書き換えれば

$$\dot{x}_p = \begin{bmatrix} \dot{x}_{p_1} \\ \dot{x}_{p_2} \\ \vdots \\ \dot{x}_{p_m} \end{bmatrix} = A_p x_p + B_p u_p = \begin{bmatrix} A_{p_1} & 0 & \cdots & 0 \\ 0 & A_{p_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_{p_m} \end{bmatrix} \begin{bmatrix} x_{p_1} \\ x_{p_2} \\ \vdots \\ x_{p_m} \end{bmatrix} + \begin{bmatrix} b_{p_1} & 0 & \cdots & 0 \\ 0 & b_{p_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & b_{p_m} \end{bmatrix} \begin{bmatrix} u_{p_1} \\ u_{p_2} \\ \vdots \\ u_{p_m} \end{bmatrix} \tag{4.20}$$

$$y_p = \begin{bmatrix} y_{p_1} \\ y_{p_2} \\ \vdots \\ y_{p_m} \end{bmatrix} = C_p x_p = \begin{bmatrix} c_{p_1}^T & 0 & \cdots & 0 \\ 0 & c_{p_2}^T & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & c_{p_m}^T \end{bmatrix} \begin{bmatrix} x_{p_1} \\ x_{p_2} \\ \vdots \\ x_{p_m} \end{bmatrix} \tag{4.21}$$

となる. ただし,

$$\begin{aligned} & x_p \in R^{n_p \times 1}, n_p \geq n_m \geq m \\ & u_p, y_p \in R^{m \times 1} \\ & A_p \in R^{n_p \times n_p}, B_p \in R^{n_p \times m}, C_p \in R^{m \times n_p} \end{aligned}$$

である. また,

$$x_{p_i}, y_{p_i}, u_{p_i} \qquad (i = 1, \cdots, m)$$

は, それぞれ適切な大きさをもつベクトルである.

この時, プラントの伝達関数行列は

$$\begin{aligned} G_p &= C_p (sI - A_p)^{-1} B_p \\ &= \begin{bmatrix} c_{p_1}^T (sI_{n_{p_1}} - A_{p_1})^{-1} b_{p_1} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & c_{p_m}^T (sI_{n_{p_m}} - A_{p_m})^{-1} b_{p_m} \end{bmatrix} \end{aligned} \tag{4.22}$$

となる.

4.5.3 適応ゲイン行列

追従誤差を

$$\begin{aligned} e_1 &= y_{p_1} - y_{m_1} \\ e_2 &= y_{p_2} - y_{m_2} \\ &\vdots \\ e_m &= y_{p_m} - y_{m_m} \end{aligned} \tag{4.23}$$

とおいて, プラントの各サブ・システムの入力信号は

$$\begin{aligned} u_{p_1} = K_1 z_1 &= K_{e_1} e_1 + K_{x_1} x_{m_1} + K_{u_1} u_{m_1} \\ u_{p_2} = K_2 z_2 &= K_{e_2} e_2 + K_{x_2} x_{m_2} + K_{u_2} u_{m_2} \\ &\vdots \\ u_{p_m} = K_m z_m &= K_{e_m} e_m + K_{x_m} x_{m_m} + K_{u_m} u_{m_m} \end{aligned} \tag{4.24}$$

となる. 一方, MIMO-SAC 論法 (第 3 章) より,

$$\begin{aligned} u_p &= Kz = \begin{bmatrix} K_{11} & K_{12} & \cdots \\ K_{21} & K_{22} & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} K_{e11} & \cdots & K_{e1m} & K_{x11} & \cdots & K_{x1m} & K_{u11} & \cdots & K_{u1m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ K_{em1} & \cdots & K_{emm} & K_{xm1} & \cdots & K_{xmm} & K_{um1} & \cdots & K_{umm} \end{bmatrix} \\ &\quad \times [\, e_1 \ \cdots \ e_m \ \mid \ x_{m_1} \ \cdots \ x_{m_m} \ \mid \ u_{m_1} \ \cdots \ u_{m_m} \,]^T \end{aligned} \tag{4.25}$$

なので, (4.23) および (4.25) より

$$K_{ij} = \begin{cases} K_i & i = j \\ 0 & i \neq j \end{cases} \tag{4.26}$$

であることが分かる. つまり,

$$K = \begin{bmatrix} K_{e_1} & 0 & \cdots & 0 & K_{x_1} & 0 & \cdots & 0 & K_{u_1} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots & 0 & \ddots & \ddots & \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & K_{e_m} & 0 & \cdots & 0 & K_{x_m} & 0 & \cdots & 0 & K_{u_m} \end{bmatrix} \tag{4.27}$$

とすれば良い.

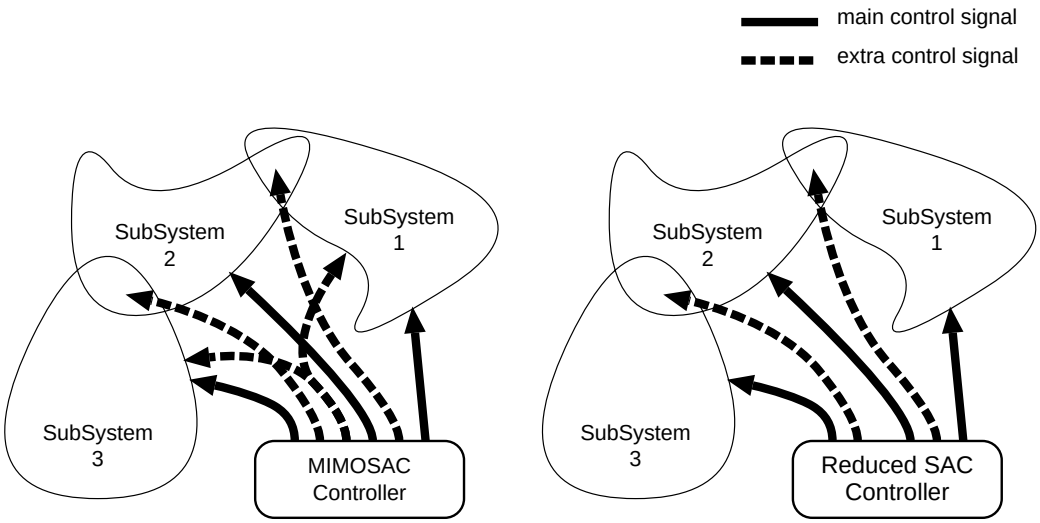


図 4.2: 直列システム

4.6 直列・連鎖的なシステム

サブシステム間に干渉項が存在する場合, システム内に信号の流れが複雑になり, SISO-SAC コントローラを並列に施して, そのまま適用すると収束性に様々な疑問が湧いてくるのであろう.

ところが, 実世界には, 大規模なシステムにおいて, そのサブシステムとサブシステムが直列につながっている場合が多い. たとえば, 多自由度ロボットマニピュレータの場合, 第 1 アームと第 2 アームが接続されているが, 第 1 アームと第 4 アームもつながっていると言うことは極めてまれなケースであろう.

このようなシステムに対して, 次の方法は有効である.

補題 4.4

干渉度指数 κ $(1 \leq \kappa \leq m)$ を導入し, 同定パラメータ行列 $K(t) = [K_e(t), K_x(t), K_u(t)]$ を次のように構成する.

$$K_e(t) = \begin{bmatrix} K_{e11} & \cdots & K_{e1\kappa} & 0 \\ \vdots & \ddots & & K_{em-\kappa+1,m} \\ K_{e1\kappa} & & \ddots & \vdots \\ 0 & K_{em-\kappa+1,m} & \cdots & K_{emm} \end{bmatrix} \tag{4.28}$$

$$K_x(t) = \begin{bmatrix} K_{x11} & \cdots & K_{x1\kappa} & 0 \\ \vdots & \ddots & & K_{xm-\kappa+1,m} \\ K_{x\kappa 1} & & \ddots & \vdots \\ 0 & K_{xm,m-\kappa+1} & \cdots & K_{xmm} \end{bmatrix} \tag{4.29}$$

$$K_u(t) = \begin{bmatrix} K_{u11} & \cdots & K_{u1\kappa} & 0 \\ \vdots & \ddots & & K_{um-\kappa+1,m} \\ K_{u\kappa 1} & & \ddots & \vdots \\ 0 & K_{um,m-\kappa+1} & \cdots & K_{umm} \end{bmatrix} \tag{4.30}$$

特に, $\kappa = 1$ の時, プラントを完全非干渉系と仮定したことになる. また, プラントが完全干渉系と仮定すれば, $\kappa = m$ となり, 全ての変数に対して干渉パラメータが計算されることになる.

証明

具体例を用いて上記の補題を説明する.

4.6.1 規範モデル

規範モデルは, 干渉項が微小な場合 (4.5.1 節) と同様に, 最小型非干渉系にする.

4.6.2 プラントの記述

一般性を失うことなく, $m = 3, \kappa = 2$ である場合を考える (これを拡張すれば任意の m および κ を設定することができる).

$$\begin{aligned} \dot{x}_1 &= A_1x_1 + b_1u_1 + \quad 0 \quad + \quad k_{12}y_2 \quad + \quad 0 \quad ; y_1 = c_1^Tx_1 \\ \dot{x}_2 &= A_2x_2 + b_2u_2 + \quad k_{21}y_1 \quad + \quad 0 \quad + \quad k_{23}y_3 \quad ; y_2 = c_2^Tx_2 \\ \dot{x}_3 &= A_3x_3 + b_3u_3 + \quad 0 \quad + \quad k_{32}y_2 \quad + \quad 0 \quad ; y_3 = c_3^Tx_3 \end{aligned} \tag{4.31}$$

ここで, k_{ij} は干渉項を表している. 各サブシステムの伝達関数を求めると

$$\begin{aligned} (sI - A_1)X_1 &= b_1U_1 + 0 + k_{12}Y_2 + 0 \\ (sI - A_2)X_2 &= b_2U_2 + k_{21}Y_1 + 0 + k_{23}Y_3 \\ (sI - A_3)X_3 &= b_3U_3 + 0 + k_{32}Y_2 + 0 \\ Y_1 &= c_1^TX_1 \end{aligned} \tag{4.32}$$

$$\begin{aligned} Y_2 &= c_2^T X_2 \\ Y_3 &= c_3^T X_3 \end{aligned} \tag{4.33}$$

となり, これを整理してまとめると

$$\begin{aligned} Y_1 &= c_1^T(sI - A_1)^{-1}[b_1 U_1 + 0 + k_{12} Y_2 + 0] \\ Y_2 &= c_2^T(sI - A_2)^{-1}[b_2 U_2 + k_{21} Y_1 + 0 + k_{23} Y_3] \\ Y_3 &= c_3^T(sI - A_3)^{-1}[b_3 U_3 + 0 + k_{32} Y_2 + 0] \end{aligned} \tag{4.34}$$

$$\begin{aligned} &\begin{bmatrix} 1 & -c_1^T(sI - A_1)^{-1}k_{12} & 0 \\ -c_2^T(sI - A_2)^{-1}k_{21} & 1 & -c_2^T(sI - A_2)^{-1}k_{23} \\ 0 & -c_3^T(sI - A_3)^{-1}k_{32} & 1 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} \\ &= \begin{bmatrix} c_1^T(sI - A_1)^{-1}b_1 & 0 & 0 \\ 0 & c_2^T(sI - A_2)^{-1}b_2 & 0 \\ 0 & 0 & c_3^T(sI - A_3)^{-1}b_3 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} \end{aligned} \tag{4.35}$$

プラントの伝達関数 G_p は

$$Q = \begin{bmatrix} 1 & -c_1^T(sI - A_1)^{-1}k_{12} & 0 \\ -c_2^T(sI - A_2)^{-1}k_{21} & 1 & -c_2^T(sI - A_2)^{-1}k_{23} \\ 0 & -c_3^T(sI - A_3)^{-1}k_{32} & 1 \end{bmatrix} \tag{4.36}$$

$$P = \begin{bmatrix} c_1^T(sI - A_1)^{-1}b_1 & 0 & 0 \\ 0 & c_2^T(sI - A_2)^{-1}b_2 & 0 \\ 0 & 0 & c_3^T(sI - A_3)^{-1}b_3 \end{bmatrix} \tag{4.37}$$

とすると

$$\begin{aligned} G_p &= Q^{-1}P \\ G_p^{-1} &= P^{-1}Q \end{aligned} \tag{4.38}$$

となる. 逆システムの伝達関数 G_p^{-1} はその帯状の形を保っていることが容易に確かめられる.

$$G_p^{-1} = \begin{bmatrix} G_{p_{11}}^{-1} & G_{p_{12}}^{-1} & 0 \\ G_{p_{21}}^{-1} & G_{p_{22}}^{-1} & G_{p_{23}}^{-1} \\ 0 & G_{p_{32}}^{-1} & G_{p_{33}}^{-1} \end{bmatrix} \tag{4.39}$$

これは, 以下に述べるフィードフォワード補償の構成に重要な意味をもつ. ただし, ここに $G_{p_{ij}}^{-1}$ は,

$$\begin{aligned} Q &= \begin{bmatrix} Q_{11} & \cdots & Q_{1m} \\ \vdots & \ddots & \vdots \\ Q_{m1} & \cdots & Q_{mm} \end{bmatrix} \\ P &= \mathrm{diag}[P_i], \qquad (i = 1, \cdots, m) \end{aligned}$$

とおいた時,

$$G_{p_{ij}}^{-1} = P_i^{-1} Q_{ij}$$

とする.

4.6.3 追従補償

4.6.2 節で述べたプラントに対し, モデルの出力に追従させるための補償器は

$$G_c = G_p^{-1} G_m = P^{-1} Q G_m \tag{4.40}$$

と構成できる. ただし,

$$P^{-1} = \begin{bmatrix} \frac{1}{c_1^T (sI - A_1)^{-1} b_1} & 0 & 0 \\ 0 & \frac{1}{c_2^T (sI - A_2)^{-1} b_2} & 0 \\ 0 & 0 & \frac{1}{c_3^T (sI - A_3)^{-1} b_3} \end{bmatrix} \tag{4.41}$$

$$G_m = \begin{bmatrix} {c_m}_1^T (sI - A_{m1})^{-1} b_{m1} & 0 & 0 \\ 0 & {c_m}_2^T (sI - A_{m2})^{-1} b_{m2} & 0 \\ 0 & 0 & {c_m}_3^T (sI - A_{m3})^{-1} b_{m3} \end{bmatrix} \tag{4.42}$$

である. また, プラントの入力を周波数ドメイン表記で表すと

$$\begin{aligned} U_p &= G_c U_m \\ &= \begin{bmatrix} G_{c11} & G_{c12} & 0 \\ G_{c21} & G_{c22} & G_{c23} \\ 0 & G_{c32} & G_{c33} \end{bmatrix} U_m \end{aligned} \tag{4.43}$$

というような形になる. よって, 上記の補題を適用することができる.

SAC では, G_c に当たる部分は適応ゲイン行列 K であることから,

$$K = \begin{bmatrix} K_{11} & K_{12} & 0 \\ K_{21} & K_{22} & K_{23} \\ 0 & K_{32} & K_{33} \end{bmatrix} \tag{4.44}$$

と構成できる. ∎

4.7 多重接続を含んだ一般のシステム

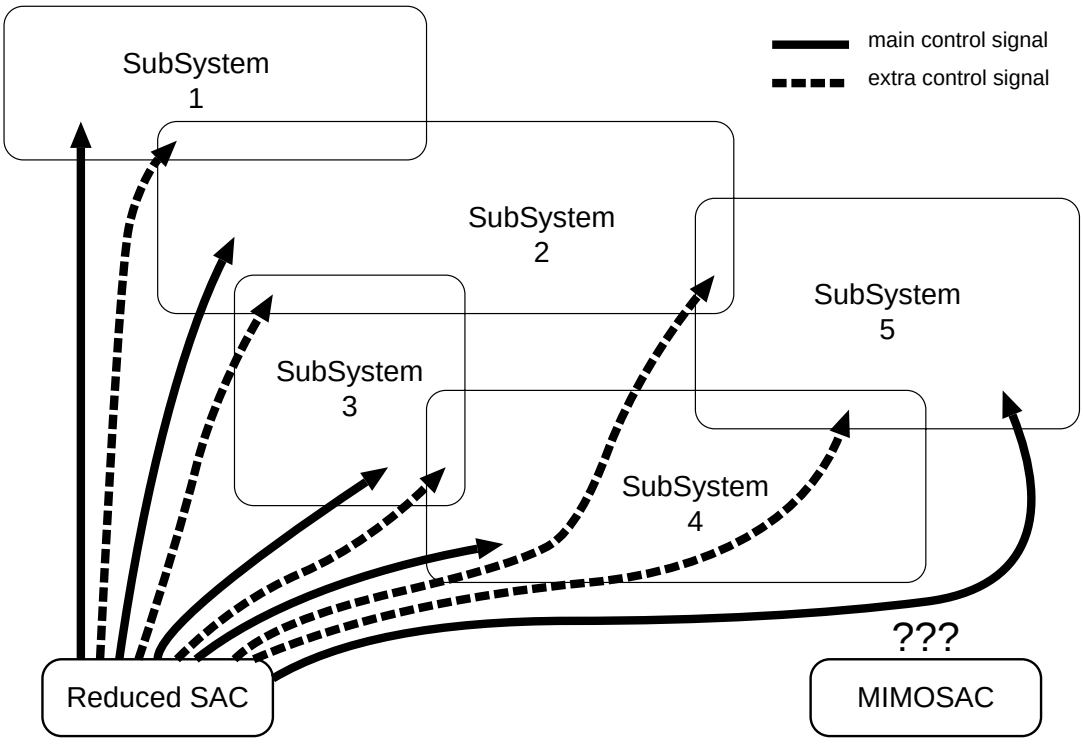


図 4.3: 多重結合を有するシステム

以上, プラントの入力に対しての干渉項しか存在しない場合, つまり特殊な場合のプラントについてコントローラゲインの設定を述べたが, 次は, より一般的な形の干渉項を持ったプラントを考えよう.

同様に, 以上の定理および補題を証明することができる.

4.7.1 規範モデル

規範モデルは, やはり干渉項が微小な場合 (4.5.1 節) と同様に, 最小型非干渉系にする.

4.7.2 制御対象の記述

一般性を失うことなく, $m = 4$ で輪型システムを考える. この場合, プラントの状態方程式は次のように表すことができる.

$$\begin{array}{l} \dot{x}_1 = A_1x_1 + b_1u_1 + \quad 0 \quad + k_{12}y_2 + \quad 0 \quad + k_{14}y_4 \\ \dot{x}_2 = A_2x_2 + b_2u_2 + k_{21}y_1 + \quad 0 \quad + k_{23}y_3 + \quad 0 \\ \dot{x}_3 = A_3x_3 + b_3u_3 + \quad 0 \quad + k_{32}y_2 + \quad 0 \quad + k_{34}y_4 \\ \dot{x}_4 = A_4x_4 + b_4u_4 + k_{41}y_1 + \quad 0 \quad + k_{43}y_3 + \quad 0 \\ y_1 = c_1^Tx_1; y_2 = c_2^Tx_2; y_3 = c_3^Tx_3; y_2 = c_2^Tx_2; y_4 = c_4^Tx_4 \end{array} \tag{4.45}$$

ここで, k_{ij} は干渉項を表している. 各サブシステムの伝達関数を求めると

$$\begin{aligned} (sI - A_1)X_1 &= b_1U_1 + 0 + k_{12}Y_2 + 0 + k_{14}Y_4 \\ (sI - A_2)X_2 &= b_2U_2 + k_{21}Y_1 + 0 + k_{23}Y_3 + 0 \\ (sI - A_3)X_3 &= b_3U_3 + 0 + k_{32}Y_2 + 0 + k_{34}Y_4 \\ (sI - A_4)X_4 &= b_4U_4 + k_{41}Y_1 + 0 + k_{43}Y_3 + 0 \end{aligned} \tag{4.46}$$

$$\begin{aligned} Y_1 &= c_1^TX_1 \\ Y_2 &= c_2^TX_2 \\ Y_3 &= c_3^TX_1 \\ Y_4 &= c_4^TX_3 \end{aligned} \tag{4.47}$$

となり, これを整理してまとめると

$$\begin{aligned} Y_1 &= c_1^T(sI - A_1)^{-1}[b_1U_1 + 0 + k_{12}Y_2 + 0 + k_{14}Y_4] \\ Y_2 &= c_2^T(sI - A_2)^{-1}[b_2U_2 + k_{21}Y_1 + 0 + k_{23}Y_3 + 0] \\ Y_3 &= c_3^T(sI - A_3)^{-1}[b_3U_3 + 0 + k_{32}Y_2 + 0 + k_{34}Y_4] \\ Y_4 &= c_4^T(sI - A_4)^{-1}[b_4U_4 + k_{41}Y_1 + 0 + k_{43}Y_3 + 0] \end{aligned} \tag{4.48}$$

$$\begin{bmatrix} 1 & -c_1^T(sI-A_1)^{-1}k_{12} & 0 & -c_1^T(sI-A_1)^{-1}k_{14} \\ -c_2^T(sI-A_2)^{-1}k_{21} & 1 & -c_2^T(sI-A_2)^{-1}k_{23} & 0 \\ 0 & -c_3^T(sI-A_3)^{-1}k_{32} & 1 & -c_3^T(sI-A_3)^{-1}k_{34} \\ -c_4^T(sI-A_4)^{-1}k_{41} & 0 & -c_4^T(sI-A_4)^{-1}k_{43} & 1 \end{bmatrix}$$
$$\times \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} c_1^T(sI-A_1)^{-1}b_1 & 0 & 0 & 0 \\ 0 & c_2^T(sI-A_2)^{-1}b_2 & 0 & 0 \\ 0 & 0 & c_3^T(sI-A_3)^{-1}b_3 & 0 \\ 0 & 0 & 0 & c_4^T(sI-A_4)^{-1}b_4 \end{bmatrix}$$
$$\times \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{4.49}$$

プラントの伝達関数 G_p は

$$Q = \tag{4.50}$$
$$\begin{bmatrix} 1 & -c_1^T(sI-A_1)^{-1}k_{12} & 0 & -c_1^T(sI-A_1)^{-1}k_{14} \\ -c_2^T(sI-A_2)^{-1}k_{21} & 1 & -c_2^T(sI-A_2)^{-1}k_{23} & 0 \\ 0 & -c_3^T(sI-A_3)^{-1}k_{32} & 1 & -c_3^T(sI-A_3)^{-1}k_{34} \\ -c_4^T(sI-A_4)^{-1}k_{41} & 0 & -c_4^T(sI-A_4)^{-1}k_{43} & 1 \end{bmatrix}$$
$$P = \begin{bmatrix} c_1^T(sI-A_1)^{-1}b_1 & 0 & 0 & 0 \\ 0 & c_2^T(sI-A_2)^{-1}b_2 & 0 & 0 \\ 0 & 0 & c_3^T(sI-A_3)^{-1}b_3 & 0 \\ 0 & 0 & 0 & c_4^T(sI-A_4)^{-1}b_4 \end{bmatrix} \tag{4.51}$$

とすると, 4.6.2 節と同様に

$$\begin{aligned} G_p &= Q^{-1}P \\ G_p^{-1} &= P^{-1}Q \end{aligned} \tag{4.52}$$

となる. 逆システムの伝達関数 G_p^{-1} は次のような構造となっている.

$$G_p^{-1} = \begin{bmatrix} G_{p11}^{-1} & G_{p12}^{-1} & 0 & G_{p14}^{-1} \\ G_{p21}^{-1} & G_{p22}^{-1} & G_{p23}^{-1} & 0 \\ 0 & G_{p32}^{-1} & G_{p33}^{-1} & G_{p34}^{-1} \\ G_{p41}^{-1} & 0 & G_{p43}^{-1} & G_{p44}^{-1} \end{bmatrix} \tag{4.53}$$

4.7.3 追従補償

4.7.2 節で述べたプラントに対し, モデルの出力に追従させるための補償器は

$$G_c = G_p^{-1} G_m = P^{-1} Q G_m \tag{4.54}$$

と構成できる. ただし,

$$P^{-1} = \begin{bmatrix} \frac{1}{c_1^T (sI - A_1)^{-1} b_1} & 0 & 0 & 0 \\ 0 & \frac{1}{c_2^T (sI - A_2)^{-1} b_2} & 0 & 0 \\ 0 & 0 & \frac{1}{c_3^T (sI - A_3)^{-1} b_3} & 0 \\ 0 & 0 & 0 & \frac{1}{c_4^T (sI - A_4)^{-1} b_4} \end{bmatrix} \tag{4.55}$$

$$G_m = \tag{4.56}$$

$$\begin{bmatrix} c_{m1}^T (sI - A_{m1})^{-1} b_{m1} & 0 & 0 & 0 \\ 0 & c_{m2}^T (sI - A_{m2})^{-1} b_{m2} & 0 & 0 \\ 0 & 0 & c_{m3}^T (sI - A_{m3})^{-1} b_{m3} & 0 \\ 0 & 0 & 0 & c_{m4}^T (sI - A_{m4})^{-1} b_{m4} \end{bmatrix}$$

である. また, プラントの入力を周波数ドメイン表記で表すと

$$\begin{aligned} U_p &= G_c U_m \\ &= \begin{bmatrix} G_{c11} & G_{c12} & 0 & G_{c14} \\ G_{c21} & G_{c22} & G_{c23} & 0 \\ 0 & G_{c32} & G_{c33} & G_{c34} \\ G_{c41} & 0 & G_{c43} & G_{c44} \end{bmatrix} U_m \end{aligned} \tag{4.57}$$

というような形になる. よって, 上記の補題を適用することができる.

SAC では, G_c に当たる部分は適応ゲイン行列 K であることから,

$$K = \begin{bmatrix} K_{11} & K_{12} & 0 & K_{14} \\ K_{21} & K_{22} & K_{23} & 0 \\ 0 & K_{32} & K_{33} & K_{34} \\ K_{41} & 0 & K_{43} & K_{44} \end{bmatrix} \tag{4.58}$$

と構成できる. ∎

4.8 シミュレーションによる検討

以上のことを数値シミュレーションによって検証してみる.

規範モデル

以下の全てのプラントに対して次のような規範モデルを設定する.

$$\begin{aligned} \dot{x}_m(t) &= A_m x_m(t) + B_m u_m(t) \\ y_m(t) &= C_m x_m(t) \\ A_m &= -I_m \\ B_m &= C_m = I_m \\ G_m(s) &= \frac{1}{s+1} I_m \end{aligned}$$

制御目的は, プラントの出力をこの規範モデルの出力の追従させることである.

制御対象

プラントの状態方程式を次のものとする (4 入力 4 出力).

$$\begin{aligned} \dot{x}_p(t) &= A_p x_p(t) + B_p u_p(t) \\ y_p(t) &= C_p x_p(t) \\ B_p &= C_p = I_4 \end{aligned}$$

- 分離型システム

$$A_p = \begin{bmatrix} 3 & 0.001 & 0.0001 & 0 \\ 0.0005 & 2 & 0 & 0 \\ 0 & 0.0002 & 4 & 0.0004 \\ 0.00003 & 0 & 0 & 1 \end{bmatrix}$$

- 直列システム

$$A_p = \begin{bmatrix} 3 & -3 & 0 & 0 \\ -1 & 2 & 3 & 0 \\ 0 & 2 & 4 & -2 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

- 輪型システム

$$A_p = \begin{bmatrix} 3 & -3 & 0 & 0 \\ -1 & 2 & 3 & 0 \\ 0 & 2 & 4 & -2 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

シミュレーション結果

以上のそれぞれの場合について数値シミュレーションの結果を示す. (図 4.4 〜 図 4.9) から分かるように, 標準 MIMOSAC と比較して簡略型 SAC のプラント出力はそれほどの差が見られない. 変数の数が少ない場合, 簡略型 SAC のプラント出力は標準 MIMOSAC より少し反応が遅いと思われるかも知れないが, 図 4.4 〜 図 4.9 は, 入力の数および出力の数が 4 であるプラントに対してサンプリング時間を 0.01 s と設定したときに, 制御プログラムのループを 4000 周期分回したシミュレーションの実行結果である. 実際, 制御ループを 100 000 周期分回すのにかかった実行時間は, 標準の MIMOSAC がおよそ 21 秒であったのに対して, 簡略型 SAC ではわずか 12 秒, つまり, ほぼ半分の時間で制御入力信号の計算が終っている, ということになる. すなわち, 計算速度が向上し, 性能的にはむしろ良くなる.

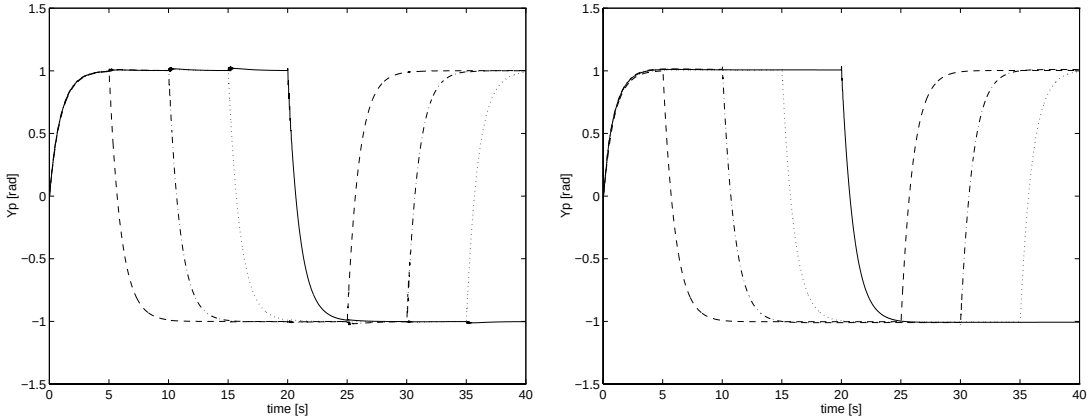
表 4.2: 簡略型 SAC の計算量

個数	掛算器	加算器	積分器
規範モデル	$2m$	m	m
適応同定	$\mathrm{O}(m\kappa)$	$\mathrm{O}(m\kappa)$	$\mathrm{O}(m\kappa)$

後述の第 6 にも示されているが, システムの規模が大きくなるとこのアルゴリズムが威力をさらに発揮するようになる. 多関節ロボットアーム制御に適用したときに, 制御関節が 5 つであきらめざるを得なかった MIMOSAC に対して, 簡略型 SAC を用いた場合, 超多関節ロボットのアーム制御にも適用の可能性が十分にあることが分かる.

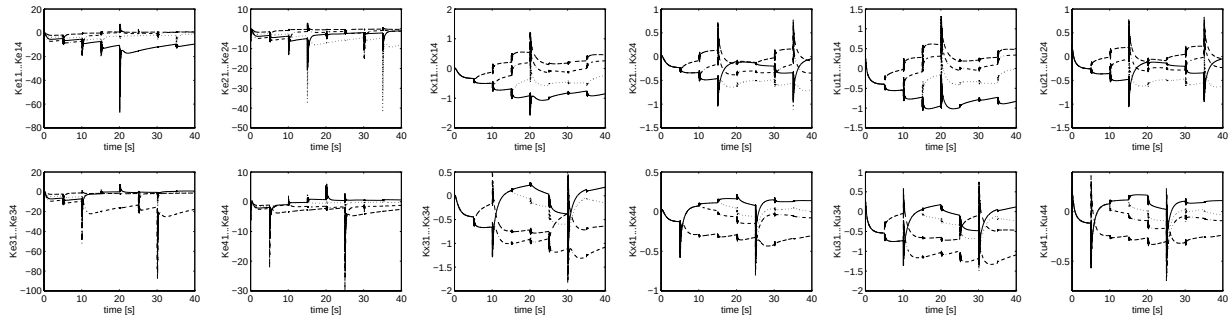
また, 表 4.2 は, 簡略型 SAC で用いられている計算器セル (積分器, 掛け算器, 加算器) の数を示している. 但し, 入力の数は出力の数と同じ $(= m)$ であるとし, κ は, メーンコントローラ以外に干渉項を補償するためにどの程度のコントローラが使われているかを表している.

なお, これらの数値シミュレーションのプログラムソースを付録に示す (msacsim2.c および msacsim3.c).

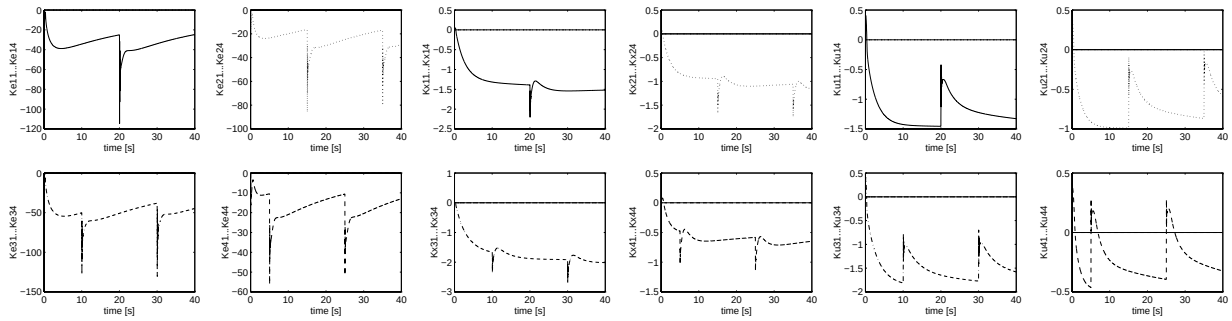


(a) MIMO-SAC (b) 簡略型 SAC

図 4.4: 4 入出力分離近似型プラント出力

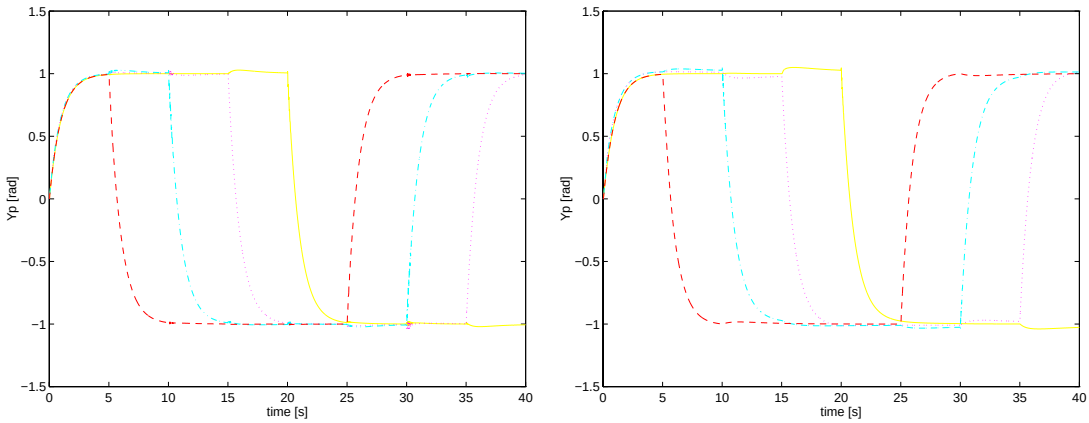


(a) MIMO-SAC



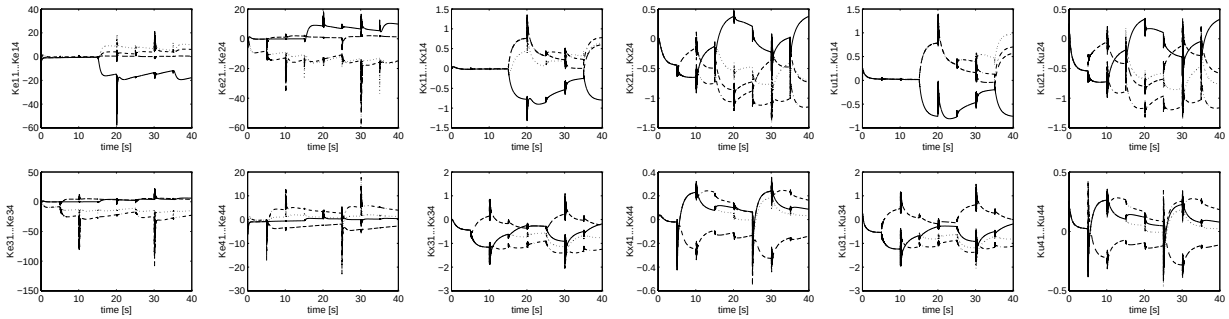
(b) 簡略型 SAC

図 4.5: 4 入出力分離型プラント適応ゲイン行列

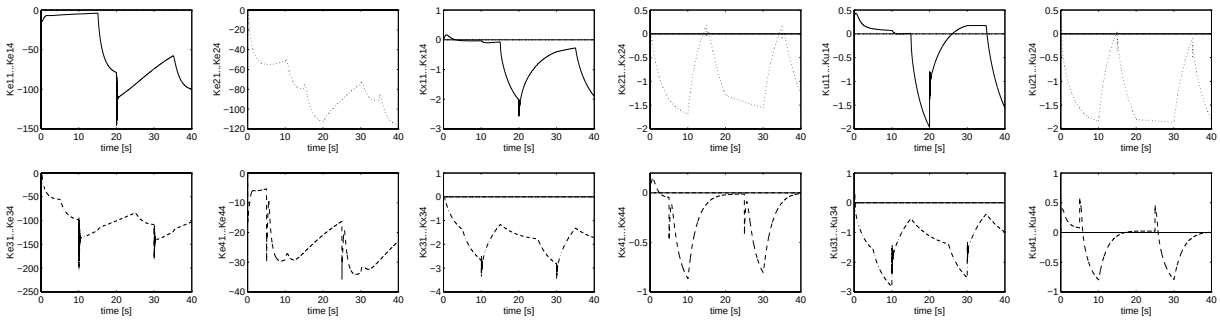


(a) MIMO-SAC (b) 簡略型 SAC

図 4.6: 4 入出力直列型プラント出力

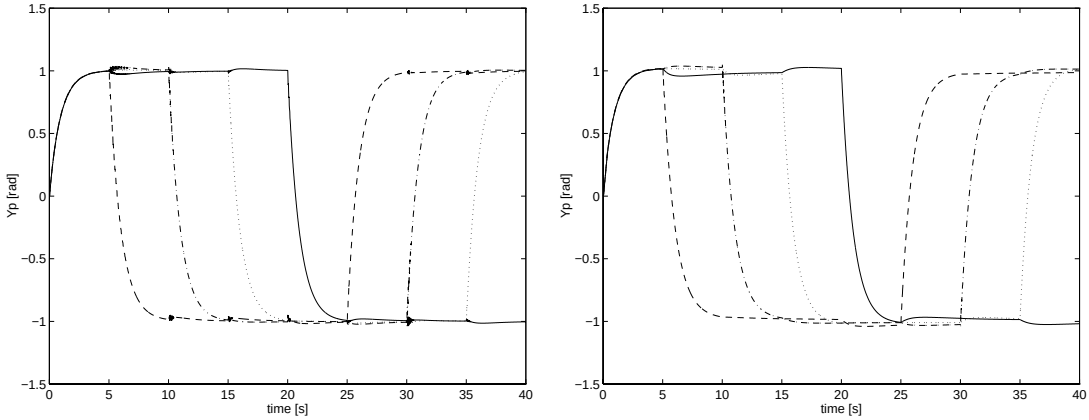


(a) MIMO-SAC



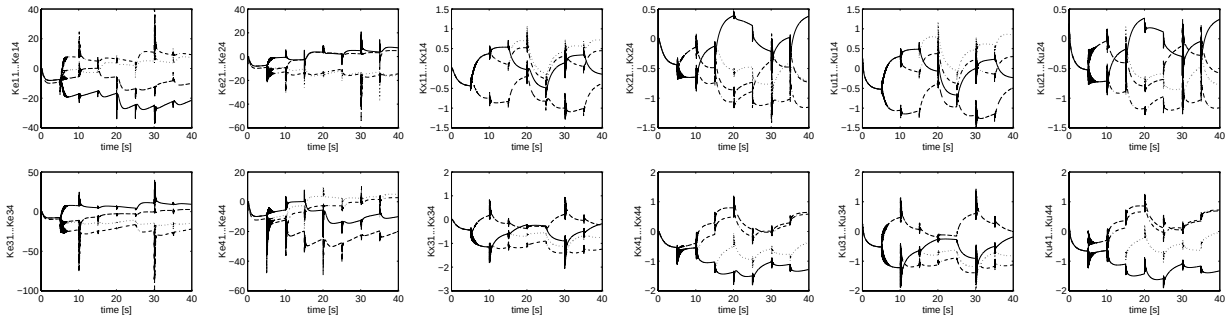
(b) 簡略型 SAC

図 4.7: 4 入出力直列型プラント適応ゲイン行列

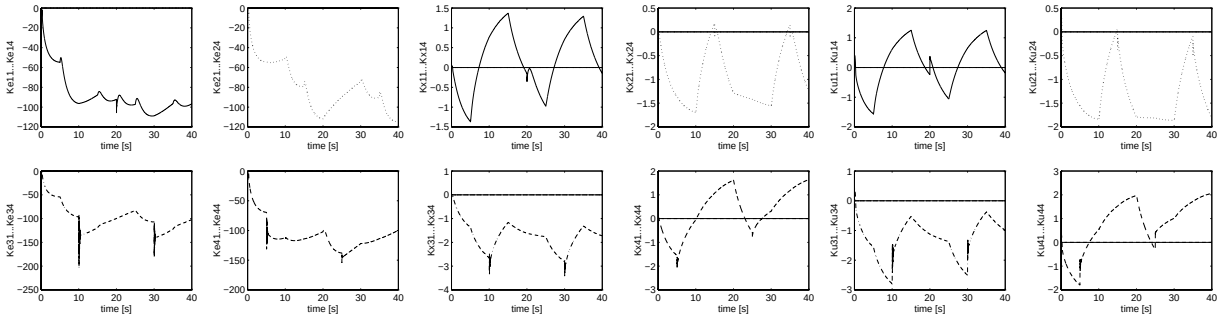


(a) MIMO-SAC (b) 簡略型 SAC

図 4.8: 4 入出力輪型プラント出力



(a) MIMO-SAC



(b) 簡略型 SAC

図 4.9: 4 入出力輪型プラント適応ゲイン行列

5 ロボットのための単純適応制御

5.1 はじめに

適応制御理論には方法論的に異なる二つの流れがある．一つがモデル参照型適応制御系 (MRACS = Model Reference Adaptive Control System) であり, もう一つは自己調整適応制御 (STR = Self-Tuning adaptive Regulator) である．それぞれの方式に基づく設計理論は, 線形システムの範囲ではほとんど完成の域に達しているとも過言ではないが, ロボット[1]の場合はダイナミクスが非線形になり, 関節変数間に動的干渉が存在するので, 一般的な線形システムに対して開発されたこれらの手法をそのまま適用しても, うまくいくかどうか保証の限りではない．

ここでは, 如何に ASPR な線形プラントに対してのみ適用できた SAC をロボット等のような非線形システムにも移植させることができるかを述べる．5.2 節および 5.3 節では SAC をロボットの制御に適用するための準備, 5.4 節, 5.5 節および 5.6 節ではその適用方法を記す．

5.2 問題設定

本来, 線形システム用に作られている単純適応制御 SAC を非線形性を含んでいるロボットに適用しようとするとさまざまなハードルが予想される．しかし, 標準の SAC に非線

[1] またはロボットマニピュレータ

形的外乱抑制機構を付き加えれば適用の可能性が十分にあると考えられる. 以下, SAC が適用可能な条件とそのための準備を述べていく.

5.2.1 非線形系制御問題

一般の非線形システムのは次のように記述できる.

$$\dot{x}(t) = f(t, x(t), u(t)) \tag{5.1}$$

その制御問題として, システムが安定となるように入力 $u(t)$ を決めることである. しかし実際, このような問題を解くのは極めて困難であり, 非線形システムに関する研究は現在世界で盛んになされているが, 妥当な解決策は未だに見つかっていない. また, ASPR の意味で可安定な線形システムに対してのみ適用可能な単純適応制御 SAC にとって, 大きな隔たりがあり, 一般の非線形システムの制御問題を解くために SAC をそのまま用いるのはかなり難しいと思われる.

ところが, 次のシステムを考えてみよう.

$$\dot{x}(t) = Ax(t) + Bu(t) + h(t, x(t)) \tag{5.2}$$

もし, 制御対象の状態方程式を線形の部分 $Ax(t) + Bu(t)$ と 非線形の部分 $h(t, x(t))$ に分けて表現できれば, しかも, 非線形項 $h(t, x(t))$ が有界であれば, SAC を適用できる可能性がある. 一入出力の場合ではあるが, 岩井ら [4] がその可能性を指摘している.

仮定 5.1

- 制御対象の状態方程式は線形の部分と非線形の部分に分けて表現できる.
- 非線形項は有界である.

5.2.2 制御対象システムの記述

ロボットの運動方程式は, 一般に

$$\{J_o + R(q)\}\ddot{q} + \frac{1}{2}\dot{R}(q)\dot{q} + S(q, \dot{q})\dot{q} + B_0\dot{q} + g(q) = Dv \tag{5.3}$$

と表せる. ここに, $v = [v_{a1}, \cdots, v_{an}]^T$ は入力電圧を表し, J_0, B_0, D は正の要素をもつ対角行列で, 次のように定義される.

$$\begin{cases} J_0 = \mathrm{diag}(J_{ii}), B_0 = \mathrm{diag}(b_{ii}), D = \mathrm{diag}(d_{ii}) \\ J_{ii} = {k_i}^2 J_i, b_{ii} = (b_{0i} + {K_i}^2/R_{ai}){k_i}^2 \\ d_{ii} = k_i K_i / R_{ai} \qquad\qquad i = 1, \cdots, n \end{cases} \tag{5.4}$$

(5.3) 式の左辺の第 1 項は慣性項であり, リンクに関する慣性行列 $R(s)$ に加えてモータのシャフトによる慣性 J_0 も考慮される. 第 2 項, 第 3 項, 第 4 項はそれぞれ遠心力, コリオリ力, および摩擦項を表す. $g(q)$ はポテンシャル項であり, q は関節変数 (一般化座標) を表している.

詳しくは参考文献 [12] を参照されたい.

5.2.3 SAC を適用するための準備

さて, 問題は, ロボットの運動方程式は (5.2) 式のように表現することができるかどうかである. そこでまず, ロボットのダイナミクス (5.3) を次の様に書き換える.

$$\ddot{q} + Q(q)f(q, \dot{q}) + Q(q)g(q) = Q(q)u \tag{5.5}$$

$$\begin{cases} q, f(q, \dot{q}), g(q) \in R^{n \times 1} \\ \qquad Q(q) \in R^{n \times n} \end{cases}$$

ここに,

$$\begin{cases} Q(q) = \{J_o + R(q)\}^{-1} \\ u = Dv \\ f(q, \dot{q}) = \frac{1}{2}\dot{R}(q)\dot{q} + S(q, \dot{q})\dot{q} + B_0\dot{q} \end{cases} \tag{5.6}$$

とおき, 遠心力とコリオリ力, 摩擦力をまとめて f で表しておいた. ここでは, 速度フィードバックは内部ループとして取り込んだうえで, 入力 u を適応制御によって決めることを考える.

さらに, 単純適応制御を適用するために, (5.2) 式の非線形項について, 次の条件を付き加える.

仮定 5.2

1. 非線形項は制御可能である. すなわち, 非線形性を補償できる外部からの入力信号が存在することである.

2. プラント (制御対象) の非線形性は検出可能である. $h(t, x(t))$ を改めて $h(t, y_h(t))$ と書き直し, $y_h(t) = Ex(t)$ は可観測な変数であり, $h(t, y_h(t))$ はこれに依存する.

ロボットの運動方程式 (5.5) 式を次の連立方程式

$$\begin{aligned} \dot{x_p}(t) &= A_p x_p(t) + B_p u_p(t) + h(t, y_h(t)) \\ y_p(t) &= C_p x_p(t), \qquad u_p, y_p \in R^{m\times 1}, m = n \\ y_h(t) &= E_p x_p(t), \qquad y_h \in R^{n_h \times 1}, n_h \leq n_p = 2n \end{aligned} \tag{5.7}$$

と変形する. 但し,

$$\begin{cases} x_p(t) = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, u_p(t) = u \\ h(t, y_h(t)) = \begin{bmatrix} 0 \\ -Q(q)(f(q, \dot{q}) + g(q)) \end{bmatrix} \\ A_p = \begin{bmatrix} 0 & I_n \\ 0 & 0 \end{bmatrix}, B_p = \begin{bmatrix} 0 \\ Q \end{bmatrix}, C_p = [\, I_n \quad 0 \,] \\ E_p = [I_{n_h} | 0] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{cases} \tag{5.8}$$

である. また $y_h(t)$ は可観測な変数とする. ここに, 次の仮定をおいている.

仮定 5.3

$$|R_{ii}(q)| \ll |J_i| \tag{5.9}$$

すなわち, 非線形的慣性項は線形な部分に比べて小である.

u_p には $Q(q) = \{J_0 + R(q)\}^{-1}$ という非線形的な要素が含まれているが, これは慣性行列を表し 対称正定 の形をしており, 後に述べる適応調整則によって補償することができる. しかし実際, 実用されている多関節型の産業ロボットの場合, J_i は $r_{ii} = R_{ii}(q)$ に比べて 10 倍以上となっているのが普通である. こうして産業ロボットは過剰設計されていると言われるが, そのおかげで運動方程式の中で非線形項があまり効いていないことになる [12]. 今回, $R(q)$ は十分小さいと考え, 話を進めることにする.

5.3 ASPR 条件の検討

5.3.1 プラント状態方程式の検証

SAC を適用するために, まず, プラント (制御対象) 状態方程式の線形部分は ASPR 条件を満たさなければならない. 線形部について, 次の仮定をおき, 非線形部については, 後に述べる適応調整メカニズムにより, 非線形性を補償することを図るものとする.

仮定 5.4

- $h(t, y_h(t)) \equiv 0$ で (5.7) の線形部および規範モデルは SAC の適用条件を満足する.

変数が 1 ($n = 1$) の場合 (5.7) 式の伝達関数は

$$G_p = \frac{1}{s^2} Q \tag{5.10}$$

となることが分かる[2]. 相対次数が 2, すなわち Zeheb の十分条件を満たしていない. この非 ASPR なプラントに対し, 次のようなフィードフォワード補償器を施す.

5.3.2 フィードフォワード補償機構の取り付け

3 章で述べたように, (5.7) 式にフィードフォワード補償を施すことにより, その拡張系を ASPR にできる. 一般に, フィードフォワード補償要素がプロパーであれば, プラントの相対次数のいかんに係わらず ASPR 可能となる[3]. フィードフォワード補償特性を状態方程式の形で

$$\begin{aligned} \dot{x}_f(t) &= A_f x_f(t) + B_f u_p(t) \\ y_f(t) &= C_f x_f(t) + D_f u_p(t) \end{aligned} \tag{5.11}$$

フィードフォワード補償機構は直達項を含んでも良いが, 漸近安定なシステムをつくるために相対次数を 1 にさせた方が良いと考えられ, ここでは $D_f = 0$ とし, ASPR 化を図

[2] $n > 1$ の場合でも $G_p = \frac{1}{s^2} I_m Q$.

[3] 具体的に, $G_p = \frac{B(s)}{A(s)} = \frac{\sum_{i=0}^{m} b_i s^i}{\sum_{i=0}^{n} a_i s^i}$, $n - m \geq 2$ で $G_f = \frac{\sum_{i=0}^{r} d_i s^i}{\sum_{i=0}^{p} c_i s^i}$, $p - r = 1$ を付加すれば $G_a = G_p + G_f$ の相対次数は $(n + p) - \max(m + p, n + r) = p - r = 1$ となる.

るものとする. ここに, $x_f(t) \in R^{n_f \times 1}$ は適切な次数の状態ベクトルである. (5.7), (5.8), (5.11) 式から, 次の拡張系 (augmented system) を得る.

$$\begin{aligned} \dot{x_a}(t) &= A_a x_a(t) + B_a u_a(t) + h_a(t, y_h(t)) \\ y_a &= C_a x_a(t) \end{aligned} \tag{5.12}$$

但し,

$$\begin{aligned} h_a(t, y_h(t)) &= \begin{bmatrix} h(t, y_h(t)) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \underline{-Q(f(q, \dot{q}) + g(q))} \\ 0 \end{bmatrix} \\ h_a(t, y_h(t)) &\in R^{n_a \times 1}, n_a = n_p + n_f \end{aligned} \tag{5.13}$$

$$\begin{cases} x_a(t) = \begin{bmatrix} x_p(t) \\ x_f(t) \end{bmatrix}, & x_a \in R^{n_a \times 1} \\ u_a(t) = u_p(t), & u_a(t), y_a(t) \in R^{m \times 1} \\ A_a = \begin{bmatrix} A_p & 0 \\ 0 & A_f \end{bmatrix}, B_a = \begin{bmatrix} B_p \\ B_f \end{bmatrix}, \bar{B} = \begin{bmatrix} 0 \\ B_f \end{bmatrix} \\ C_a = \begin{bmatrix} C_p & C_f \end{bmatrix} \end{cases} \tag{5.14}$$

具体的に, 変数が 1 の場合

$$A_f = -a, B_f = b, C_f = 1 \tag{5.15}$$

とし, このフィードフォワードの伝達関数は

$$G_f = \frac{b}{s + a} \tag{5.16}$$

であり[4], 合成システムの伝達関数は

$$G_a = G_p + G_f = \frac{s^2 b + (s + a)Q}{s^3 + s^2 a} \tag{5.17}$$

となり, D を正定に選べば, 拡張系は ASPR 条件を満たす. また, b を十分小さくしておけば元のシステムにほとんど影響がない.

この, 合成されたロボットのダイナミクス (5.12) に対して, 漸近安定な線形規範モデルを

$$\begin{aligned} \dot{x}_m(t) &= A_m x_m(t) + B_m u_m(t) \\ y_m(t) &= C_m x_m(t) \end{aligned} \tag{5.18}$$

[4] $n > 1$ の場合 $G_f = \frac{b}{s+a} I_m$

とする.

制御目的は,

$$\begin{aligned} e_y(t) &= y_a(t) - y_m(t) \\ \lim_{t\to\infty} e_y(t) &= 0 \end{aligned} \tag{5.19}$$

を実現することである.

$h_a(t, y_h(t)) \equiv 0$ として合成システム (5.12) が規範モデルを完全に追従したとき, $x_a^*(t)$, $u_a^*(t)$, $y_a^*(t)$ をその時のプラント状態量, および入出力とおくと, 次の式が成り立つ.

$$\dot{x}_a^*(t) = A_a x_a^*(t) + B_a u_a^*(t) \tag{5.20}$$

$$y_m(t) = y_a^*(t) = C_a x_a^*(t) \tag{5.21}$$

また, プラントと規範モデルの状態量および入力項の間に次の関係が成立する.

$$x_a^*(t) = S_{11} x_m(t) + S_{12} u_m(t) + S_{13}(t) \tag{5.22}$$

$$u_a^*(t) = S_{21} x_m(t) + S_{22} u_m(t) + S_{23}(t) \tag{5.23}$$

5.4 コントローラの設計

5.4.1 制御入力の選定

さて, 実際にはプラントのパラメータは未知なので, (5.22),(5.23) 式を構成することはできない.

その時の実際の状態と理想状態とのずれを

$$\begin{aligned} \Delta x_a(t) &= x_a(t) - x_a^*(t) \\ e_y(t) &= y_a(t) - y_m(t) = y_a(t) - y_a^*(t) \\ &= C_a(x_a(t) - x_a^*(t)) = C_a \Delta x_a(t) \end{aligned} \tag{5.24}$$

とおく. ここで, 制御入力を

$$u_p(t) = u_a(t) = K(t) z(t) + u_R(t) \tag{5.25}$$

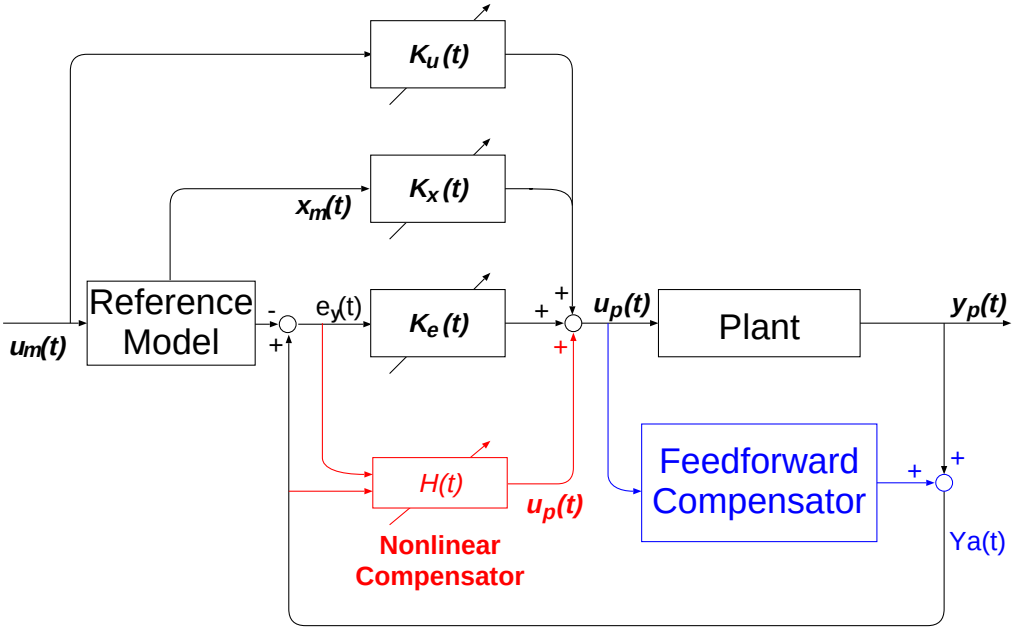


図 5.1: ロボット用 SAC の構成図

とする. 但し,

$$K(t) = [K_e(t), K_x(t), K_u(t)] \tag{5.26}$$

$$z(t) = [e_y(t)^T, x_m(t)^T, u_m(t)^T]^T \tag{5.27}$$

また, 非線形外乱抑制入力 $u_R(t) = [u_{R1}(t), \cdots, u_{Rm}(t)]^T$ は

$$H_i(t) = [\rho_{0i},\ \rho_{1i}] \qquad \rho_{0i}, \rho_{1i} > 0$$

$$z_h(t) = \begin{bmatrix} 1 \\ ||y_h(t)|| \end{bmatrix}$$

$$e_y(t) = \begin{bmatrix} e_{y_1}(t) \\ \vdots \\ e_{y_m}(t) \end{bmatrix}$$

とするとき,

$$|u_{Ri}(t)| \leq H_i(t) z_h(t)$$

$$u_{Ri}(t) = \begin{cases} -H_i(t) z_h(t)\ \mathrm{sgn}\{e_{y_i}(t)\}, & |H_i(t) z_h(t) e_{y_i}(t)| > \epsilon \\ -\{H_i(t) z_h(t)\}^2 e_{y_i}(t)/\epsilon, & |H_i(t) z_h(t) e_{y_i}(t)| \leq \epsilon \end{cases} \tag{5.28}$$

と与える. 但し, ϵ は微小正定数である.

(5.12), (5.25), および (5.20) 〜 (5.23) 式から

$$\begin{aligned}
\Delta K(t) &= [K_e(t) - K_e^*, K_x(t) - S_{21}, K_u(t) - S_{22}] && (5.29)\\
\Delta \dot{x}_a(t) &= A_a \Delta x_a(t) + B_a(u_a(t) - u_a^*(t)) + h_a(t, y_h(t)) && (5.30)\\
&= A_a \Delta x_a(t) + B_a(\underline{\Delta K(t) z(t) + u_R(t) - S_{23}(t)} + K_e^* e_y(t))\\
&\quad + h_a(t, y_h(t))\\
&= A_a \Delta x_a(t) + B_a(u_{ac} + K_e^* C_a \Delta x_a(t)) + h_a(t, y_h(t))\\
&= (\underline{A_a + B_a K_e^* C_a}) \Delta x_a(t) + B_a u_{ac} + h_a(t, y_h(t))\\
&= A_{ac} \Delta x_a(t) + B_a u_{ac}(t) + h_a(t, y_h(t)) && (5.31)\\
u_{ac}(t) &= \Delta K(t) z(t) + u_R(t) - S_{23}(t) && (5.32)
\end{aligned}$$

を得る. SAC 構成法の仮定 2.1 および仮定 3.1 より (5.31) 式の線形部は, SPR であることから, Kalman-Yakubovich の補題より,

$$\begin{aligned}
{A_{ac}}^T P + P A_{ac} &= -Q_{ac} && (5.33)\\
{B_a}^T P &= C_a
\end{aligned}$$

を満足する正定対称行列 $P = P^T > 0, Q_{ac} = Q_{ac}^T > 0$ が存在する.

(5.31) 式において,

$$\lim_{t \to \infty} B_a u_{ac} + h_a(t, y_h(t)) = 0 \qquad (5.34)$$

であればシステムが漸近安定 ($\lim_{t \to \infty} e_y(t) = 0$) となる. つまり, システムが漸近安定となるように $u_{ac}(t)$ を求めれば良い. その一つの方法として,

$$\begin{aligned}
&\lim_{t \to \infty} \Delta K(t) z(t) = 0 && (5.35)\\
&\lim_{t \to \infty} B_a(u_R(t) - S_{23}(t)) + h_a(t, y_h(t)) = 0 && (5.36)
\end{aligned}$$

となるように $u_{ac}(t)$ を決めよう. (5.36) 式において, 次の仮定が成立すれば, u_R が楽に求めることができるであろう.

仮定 5.5

1. システム (5.12) において, $h_a(t, y_h(t)) \in R^{n_a \times 1}$ に対し,

$$h_a(t, y_h(t)) = (B_a - \bar{B})\hat{h}_a(t, y_h(t)) \tag{5.37}$$

なる $\hat{h}_a(t, y_h(t)) \in R^{m \times 1}$ が存在する. すなわち, $h(t, y_h(t)) \in R^{n_p \times 1}$ に対し,

$$h(t, y_h(t)) = B_p\hat{h}(t, y_h(t)) \tag{5.38}$$

なる $\hat{h}(t, y_h(t)) \in R^{m \times 1}$ が存在するということになる. これによって, 外部からの制御信号によって非線形性を補償することができる.

2. 非線形項 $\hat{h}(t, y_h(t)) = [\hat{h}_1(t, y_h(t)), \cdots, \hat{h}_m(t, y_h(t))]^T$ は次の意味で有界である.

$$|\hat{h}_i(t, y_h(t))| \leq \rho_{0i} + \rho_{1i}||y_h(t)|| \tag{5.39}$$

となる定数 $\rho_{0i}, \rho_{1i} > 0$ が存在する. 非線形性抑制機構は, これらの上限値 (ρ_0 および ρ_1) を用い, 非線形項を補償する.

これらをロボットのダイナミックス (5.7) 〜 (5.14) において検証してみよう. (5.38) が成立していることはすぐに確認できるのであろう. (5.8) より,

$$\hat{h}(t, y_h(t)) = -(f(q, \dot{q}) + g(q)) \tag{5.40}$$

となることが分かる. また, これについて $f(q, \dot{q}) + g(q)$ が有界であれば (5.39) が成立することになる.

5.4.2 Lyapunov 法による適応調整則の選定

(5.32) 式において, 条件 (5.34) 式を満たす $K(t)$ および $u_R(t)$ さえ見つければ良いので, 次のスカラー正定 Lyapunov 関数を考える[5].

$$\begin{aligned} V(t) \;=\; & \Delta x_a(t)^T P \Delta x_a(t) \\ & + \mathrm{trace}\{\Delta K(t) \Gamma_K^{-1} \Delta K(t)^T\} \\ & + \sum_{i=1}^{m} \Delta H_i(t) (\Gamma_H)_i^{-1} \Delta H_i(t)^T \end{aligned} \tag{5.41}$$

[5]求めるパラメータ ($\Delta K(t)$ と $\Delta H(t)$) とその解を持つと思われる候補 $\Delta \dot{x}_a(t)$ を含むように Lyapunov 関数を設計する.

但し,

$$\Delta K(t) = K(t) - K^* \tag{5.42}$$

$$\Delta H_i(t) = H_i(t) - H_i^* \tag{5.43}$$

とおいている.

(5.41) から微分をとり, (5.42) 〜 (5.43) より

$$\begin{aligned}\dot{V}(t) &= \Delta\dot{x}_a(t)^T P\Delta x_a(t) + \Delta x_a(t)^T P\Delta\dot{x}_a(t) \\ &\quad +2\left\{\mathrm{trace}\{\Delta\dot{K}(t)\Gamma_K^{-1}\Delta K(t)^T\} + \sum_{i=1}^{m}\Delta\dot{H}_i(t)(\Gamma_H)_i^{-1}\Delta H_i(t)^T\right\}\end{aligned} \tag{5.44}$$

を得る. 一方, (5.44) 式の右側の第 1 項は

$$\begin{aligned}&\Delta\dot{x}_a(t)^T P\Delta x_a(t) + \Delta x_a(t)^T P\Delta\dot{x}_a(t) \\ &= (A_{ac}\Delta x_a + B_a u_{ac} + h_a(t, y_h(t)))^T P\Delta x_a(t) \\ &\quad +\Delta x_a(t)^T P(A_{ac}\Delta x_a + B_a u_{ac} + h_a(t, y_h(t))) \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) + 2\left\{u_{ac}(t)^T B_a^T P\Delta x_a(t) + h_a(t, y_h(t))^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) + 2\left\{u_{ac}(t)^T C_a\Delta x_a(t) + \hat{h}_a(t, y_h(t))^T(B_a - \bar{B})^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{u_{ac}(t)^T e_y(t) + \hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{(\Delta K(t)z(t) + u_R(t) - S_{23}(t))^T e_y(t)\right. \\ &\quad \left.+\hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{\mathrm{trace}\{e_y(t)z(t)^T\Delta K(t)^T\} + (u_R(t) - S_{23}(t))^T e_y(t)\right. \\ &\quad \left.+\hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\}\end{aligned} \tag{5.45}$$

であり, 第 2 項および第 3 項は

$$\mathrm{trace}\{\Delta\dot{K}(t)\Gamma_K^{-1}\Delta K(t)^T\} = \mathrm{trace}\{\dot{K}(t)\Gamma_K^{-1}\Delta K(t)^T\} \tag{5.46}$$

$$\Delta\dot{H}_i(t)(\Gamma_H)_i^{-1}\Delta H_i(t)^T = \dot{H}_i(t)(\Gamma_H)_i^{-1}\Delta H_i(t)^T \tag{5.47}$$

であることから,

$$\begin{aligned} \hat{h}_a(t, y_h(t)) &\equiv 0 \\ S_{23}(t) &\equiv 0 \end{aligned}$$

であるとき,

$$\dot{K}(t) = -e_y(t) z(t)^T \Gamma_K \tag{5.48}$$

そうでない場合は

$$|\hat{h}_a(t, y_h(t))^T \bar{B}^T P \Delta x_a(t)| = \rho_2 \tag{5.49}$$

$$|(-S_{23i}(t) + \hat{h_{ai}}(t, y_h(t))) e_{y_i}(t)| \leq |S_{23i}| + \rho_{0i} + \rho_{1i} ||y_h(t)|| \tag{5.50}$$

$$[\, \rho_{0i} + \left|\tfrac{\rho_{2i}}{e_{y_i}(t)}\right| + |S_{23i}(t)|\,, \quad \rho_{1i}\,] \begin{bmatrix} 1 \\ ||y_h(t)|| \end{bmatrix} = z_h(t)^T H_i^{*T} \tag{5.51}$$

とおくと

$$\begin{aligned} (u_R(t) - S_{23}(t))^T e_y(t) + \hat{h}_a(t, y_h(t))^T e_y(t) &- \hat{h}_a(t, y_h(t))^T \bar{B}^T P \Delta x_a(t) \\ &= -\sum_{i=1}^{m} e_{y_i}(t) z_h(t)^T (H_i(t) - H_i^*)^T \\ &= -\sum_{i=1}^{m} e_{y_i}(t) z_h(t)^T \Delta H_i(t)^T \end{aligned} \tag{5.52}$$

すなわち

$$\dot{H}_i = |e_{y_i}(t)| z_h(t)^T \Gamma_{Hi} \tag{5.53}$$

がさらに必要となる.

こうすれば

$$\dot{V}(t) < 0 \qquad\qquad (\Delta x_a(t) \neq 0) \tag{5.54}$$

となり, システムは Lyapunov 的に安定となり, $\lim_{t\to\infty} e_y(t) = 0$ が達成される.

しかし, (5.48) は $-e_y(t)^2$ を含んでおり, これは常にマイナスの値を取っているので外乱などによる影響を受けやすい. そこで, 次のような積分比例型適応調整則を使うことに

する.

$$
\begin{aligned}
K(t) &= K_I(t) + K_P(t) && (5.55)\\
\dot{K}_I(t) &= -e_y(t)z(t)^T\Gamma_{K_I} - \sigma_K(t)K_I(t) && (5.56)\\
K_P(t) &= -e_y(t)z(t)^T\Gamma_{K_P} && (5.57)\\
H_i(t) &= H_{Ii}(t) + H_{Pi}(t) && (5.58)\\
\dot{H}_{Ii}(t) &= |e_{y_i}(t)|z_h(t)^T(\Gamma_{H_I})_i - \sigma_{Hi}(t)H_{Ii}(t) && (5.59)\\
H_{Pi}(t) &= |e_{y_i}(t)|z_h(t)^T(\Gamma_{H_P})_i \geq 0 && (5.60)
\end{aligned}
$$

$$
\begin{aligned}
\Gamma_{K_I} &= \Gamma_{K_I}^T > 0\\
\Gamma_{K_P} &= \Gamma_{K_P}^T > 0\\
(\Gamma_{H_I})_i &= (\Gamma_{H_I})_i^T > 0\\
(\Gamma_{H_P})_i &= (\Gamma_{H_P})_i^T > 0\\
\sigma_K(t) &= \sigma_{K0}\frac{e_y(t)^T e_y(t)}{1 + e_y(t)^T e_y(t)}\\
\sigma_{Hi}(t) &= \sigma_{H0i}\frac{e_{y_i}(t)^2}{1 + e_{y_i}(t)^2}\\
\sigma_{K0} &\quad , \quad \sigma_{H0i} > 0
\end{aligned}
$$

上記の Lyapunov 関数を次のように書き直す.

$$
\begin{aligned}
V(t) &= \Delta x_a(t)^T P \Delta x_a(t)\\
&\quad + \mathrm{trace}\{\Delta K_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\}\\
&\quad + \sum_{i=1}^{m}\Delta H_{Ii}(t)(\Gamma_{H_I})_i\Delta H_{Ii}(t)^T \qquad (5.61)
\end{aligned}
$$

但し,

$$
\begin{aligned}
\Delta K(t) &= K(t) - K^* = (K_I(t) + K_P(t)) - K^* && (5.62)\\
\Delta H_i(t) &= H_i(t) - H_i^* = (H_{Ii}(t) + H_{Pi}(t)) - H_i^* && (5.63)\\
\Delta K_I(t) &= K_I(t) - K^* \leq \Delta K(t) && (5.64)\\
\Delta H_{Ii}(t) &= H_{Ii}(t) - H_i^* \leq \Delta H_i(t) && (5.65)
\end{aligned}
$$

とおいている.

(5.61) から微分をとり,

$$\begin{aligned}\dot{V}(t) &= \Delta\dot{x}_a(t)^T P\Delta x_a(t) + \Delta x_a(t)^T P\Delta\dot{x}_a(t) \\ &\quad +2\left\{\mathrm{trace}\{\Delta\dot{K}_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} + \sum_{i=1}^{m}\Delta\dot{H}_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T\right\} \quad (5.66)\end{aligned}$$

を得る. 一方,

$$\begin{aligned}&\Delta\dot{x}_a(t)^T P\Delta x_a(t) + \Delta x_a(t)^T P\Delta\dot{x}_a(t) \\ &= (A_{ac}\Delta x_a + B_a u_{ac} + h_a(t, y_h(t)))^T P\Delta x_a(t) \\ &\quad +\Delta x_a(t)^T P(A_{ac}\Delta x_a + B_a u_{ac} + h_a(t, y_h(t))) \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) + 2\left\{u_{ac}(t)^T B_a^T P\Delta x_a(t) + h_a(t, y_h(t))^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) + 2\left\{u_{ac}(t)^T C_a\Delta x_a(t) + \hat{h}_a(t, y_h(t))^T(B_a - \bar{B})^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{u_{ac}(t)^T e_y(t) + \hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{(\Delta K(t)z(t) + u_R(t) - S_{23}(t))^T e_y(t)\right. \\ &\quad \left.+\hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\} \\ &= -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ &\quad +2\left\{\mathrm{trace}\{e_y(t)z(t)^T\Delta K(t)^T\} + (u_R(t) - S_{23}(t))^T e_y(t)\right. \\ &\quad \left.+\hat{h}_a(t, y_h(t))^T e_y(t) - \hat{h}_a(t, y_h(t))^T\bar{B}^T P\Delta x_a(t)\right\} \quad (5.67)\end{aligned}$$

$$\mathrm{trace}\{\Delta\dot{K}_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} = \mathrm{trace}\{\dot{K}_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} \quad (5.68)$$

$$\Delta\dot{H}_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T = \dot{H}_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T \quad (5.69)$$

であることから,

$$\begin{aligned}&\mathrm{trace}\{\dot{K}_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} + \sum_{i=1}^{m}\dot{H}_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T \\ &= \mathrm{trace}\{(-e_y(t)z(t)^T\Gamma_{K_I} - \sigma_K(t)K_I(t))\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} \\ &\quad +\sum_{i=1}^{m}\{(|e_{y_i}(t)|z_h(t)^T(\Gamma_{H_I})_i - \sigma_{Hi}H_{Ii}(t))(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T\} \\ &= \mathrm{trace}\{-e_y(t)z(t)^T\Delta K_I(t)^T - \sigma_K(t)K_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} \\ &\quad +\sum_{i=1}^{m}\{|e_{y_i}(t)|z_h(t)^T\Delta H_{Ii}(t)^T - \sigma_{Hi}H_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T\}\end{aligned}$$

$$\begin{aligned} &\leq \mathrm{trace}\{-e_y(t)z(t)^T\Delta K(t)^T - \sigma_K(t)K_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} \\ &\quad + \sum_{i=1}^{m}\{|e_{y_i}(t)|z_h(t)^T\Delta H_i(t)^T - \sigma_{Hi}H_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T\} \end{aligned} \tag{5.70}$$

また

$$\begin{aligned} &|\hat{h}_a(t,y_h(t))^T\bar{B}^TP\Delta x_a(t)| \\ &\quad \leq \lambda_{max}[P]\left(\sum_{i=1}^{m}(\rho_{0i}+\rho_{1i}||y_h(t)||)\right)\left\|\bar{B}\right\|\|\Delta x_a(t)\| \end{aligned} \tag{5.71}$$

$$\begin{aligned} &(u_R(t)-S_{23}(t))^Te_y(t) + \hat{h}_a(t,y_h(t))^Te_y(t) \\ &\quad = -\sum_{i=1}^{m}e_{y_i}(t)z_h(t)^T\Delta H_i(t)^T \end{aligned} \tag{5.72}$$

とすると結局,

$$\begin{aligned} \dot{V}(t) \leq\ & -\Delta x_a(t)Q_{ac}\Delta x_a(t) \\ & -2\ \mathrm{trace}\{\sigma_K(t)K_I(t)\Gamma_{K_I}^{-1}\Delta K_I(t)^T\} \\ & -2\sum_{i=1}^{m}\{\sigma_{Hi}H_{Ii}(t)(\Gamma_{H_I})_i^{-1}\Delta H_{Ii}(t)^T\} \\ & +2\epsilon + 2\lambda_{max}[P]\left(\sum_{i=1}^{m}(\rho_{0i}+\rho_{1i}||y_h(t)||)\right)\left\|\bar{B}\right\|\|\Delta x_a(t)\| \end{aligned} \tag{5.73}$$

となる. よって, 1 入出力の場合と同様に, 系内の全ての信号が有界であることが証明できる.

5.5 SAC のロボットへの適用

以上, 単純適応制御 (SAC) がロボットのアーム制御にも適用可能であることを証明し, そのコントローラの設計法を述べた.

5.5.1 適用条件

上記の SAC のロボットへの適用の仕方などの説明を以下にまとめる. まず, 制御対象となっているロボット・と規範モデルおよびコントローラに対して次の条件をおく.

仮定 5.6

1. ロボットの状態方程式は線形の部分と非線形の部分に分けて表現できる.

$$\dot{x}(t) = Ax(t) + Bu(t) + h(t, x(t)) \tag{5.74}$$

制御対象の状態方程式は線形の部分 $Ax(t) + Bu(t)$ と 非線形の部分 $h(t, x(t))$ に分けて表現できる.

2. ロボットの状態方程式の非線形項は有界である.

3. ロボットの状態方程式の非線形項は制御可能である. すなわち, 非線形性を補償できる外部からの入力信号が存在する. そのマッチング条件として, 非線形項 $h(t, y_h(t)) \in R^{n_p \times 1}$ に対し,

$$h(t, y_h(t)) = B_p \hat{h}(t, y_h(t)) \tag{5.75}$$

なる $\hat{h}(t, y_h(t)) \in R^{m \times 1}$ が存在することである.

4. プラント (制御対象) となるロボットの非線形性は外部から検出可能である. (5.74) において, 非線形項 $h(t, x(t))$ を改めて $h(t, y_h(t))$ と書き直し, $y_h(t) = Ex(t)$ は可観測な変数であり, $h(t, y_h(t))$ はこれに依存する.

$$\begin{aligned} \dot{x_p}(t) &= A_p x_p(t) + B_p u_p(t) + h(t, y_h(t)) \\ y_p(t) &= C_p x_p(t), \qquad u_p, y_p \in R^{m \times 1}, m = n \\ y_h(t) &= E_p x_p(t), \qquad y_h \in R^{n_h \times 1}, n_h \leq n_p = 2n \end{aligned} \tag{5.76}$$

5. $h(t, y_h(t)) \equiv 0$ で (5.76) の線形部および規範モデルは SAC の適用条件を満足する.

プラント (5.76) の線形部は ASPR 条件を満たし, 規範モデルとして漸近安定なものを選ぶ. ASPR 条件が満足されていない場合, フィードフォワード 補償機構を付加することによって ASPR 化を図るものとする.

6. フィードフォワード 補償機構 (PFC) を用いた場合, その合成システムにおいて, $h_a(t, y_h(t)) \in R^{n_a \times 1}$ に対し,

$$h_a(t, y_h(t)) = (B_a - \bar{B})\hat{h}_a(t, y_h(t))$$

なる $\hat{h}_a(t, y_h(t)) \in R^{m \times 1}$ が存在する. これによって, 外部からの制御信号によって非線形性を補償することができる.

7. 非線形項 $\hat{h}(t, y_h(t)) = [\hat{h}_1(t, y_h(t)), \cdots, \hat{h}_m(t, y_h(t))]^T$ は次の意味で有界である.

$$|\hat{h}_i(t, y_h(t))| \leq \rho_{0i} + \rho_{1i}||y_h(t)|| \tag{5.77}$$

となる定数 $\rho_{0i}, \rho_{1i} > 0$ が存在する. 非線形性補償機構は, これらの上限値 (ρ_0 および ρ_1) を用い, 非線形項を補償する.

8. ロボットの非線形的慣性項は線形な部分に比べて小である.

$$|R_{ii}(q)| \ll |J_i|$$

すなわち, 慣性項は線形項として扱うことにする.

これらの条件が ロボット において成立している (実現できる) ことを以下に記す.

5.5.2 制御対象の状態方程式 (ロボット運動方程式) の記述

ロボット の運動方程式は

$$\{J_o + R(q)\}\ddot{q} + \frac{1}{2}\dot{R}(q)\dot{q} + S(q, \dot{q})\dot{q} + B_0\dot{q} + g(q) = Dv \tag{5.78}$$

と表せる. ここに, $v = [v_{a1}, \cdots, v_{an}]^T$ はモータに対する入力電圧を表し, J_0, B_0, D は正の要素をもつ対角行列で, 次のように定義される.

$$\begin{cases} J_0 = \mathrm{diag}(J_{ii}), B_0 = \mathrm{diag}(b_{ii}), D = \mathrm{diag}(d_{ii}) \\ J_{ii} = {k_i}^2 J_i, b_{ii} = (b_{0i} + {K_i}^2/R_{ai}){k_i}^2 \\ d_{ii} = k_i K_i / R_{ai} \qquad\qquad i = 1, \cdots, n \end{cases} \tag{5.79}$$

(5.78) 式の左辺の第 1 項は慣性項であり, リンクに関する慣性行列 $R(s)$ に加えてモータのシャフトによる慣性 J_0 も入っている. 第 2 項, 第 3 項, 第 4 項はそれぞれ遠心力, コリオリ力, および摩擦項を表す. $g(q)$ はポテンシャル項であり, q は関節変数 (一般化座標) を表している.

さて, このロボットの運動方程式は (5.76) 式のように, 線形の部分と非線形の部分に分けて表現することができることを示す. まず, ロボットのダイナミクス (5.78) を次の様に書き換える.

$$\ddot{q} + Q(q)f(q, \dot{q}) + Q(q)g(q) = Q(q)u \tag{5.80}$$

$$\begin{cases} q, f(q,\dot{q}), g(q) \in R^{n\times 1} \\ \qquad Q(q) \in R^{n\times n} \end{cases}$$

ここに,

$$\begin{cases} Q(q) = \{J_o + R(q)\}^{-1} \\ u = Dv \\ f(q,\dot{q}) = \frac{1}{2}\dot{R}(q)\dot{q} + S(q,\dot{q})\dot{q} + B_0\dot{q} \end{cases} \tag{5.81}$$

とおき, 遠心力とコリオリ力, 摩擦力をまとめて f で表しておいた. ここでは, 速度フィードバックは内部ループとして取り込んだうえで, 入力 u を適応制御によって決めることを考える.

慣性行列 $Q(q)$ には $R(q)$ という非線形的な要素が含まれているが, 実際, 実用されている多関節型の産業ロボットの場合, J_i は $r_{ii} = R_{ii}(q)$ に比べて 10 倍以上となっているのが普通である. こうして産業ロボットは過剰設計されていると言われるが, そのおかげで運動方程式の中で非線形項があまり効いていないことになる. $R(q)$ は十分小さいと考え, 仮定 5.6.8 が成立しているとする. すなわち, 慣性項 $Q(q)$ は線形項 Q として扱うことにする.

したがって, ロボットの運動方程式 (5.78) 式は次の連立方程式

$$\begin{aligned} \dot{x_p}(t) &= A_p x_p(t) + B_p u_p(t) + h(t, y_h(t)) \\ y_p(t) &= C_p x_p(t), \qquad u_p, y_p \in R^{m\times 1}, m = n \\ y_h(t) &= E_p x_p(t), \qquad y_h \in R^{n_h\times 1}, n_h \le n_p = 2n \end{aligned} \tag{5.82}$$

と変形できる. 但し,

$$\begin{cases} x_p(t) = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, u_p(t) = u = Dv \\ h(t, y_h(t)) = \begin{bmatrix} 0 \\ -Q(f(q,\dot{q}) + g(q)) \end{bmatrix} \\ = \begin{bmatrix} 0 \\ -Q((\frac{1}{2}\dot{R}(q) + S(q,\dot{q}) + B_0)\dot{q} + g(q)) \end{bmatrix} \\ A_p = \begin{bmatrix} 0 & I_n \\ 0 & 0 \end{bmatrix}, B_p = \begin{bmatrix} 0 \\ Q \end{bmatrix}, C_p = [\, I_n \quad 0 \,] \\ E_p = [I_{n_h}|0] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{cases} \tag{5.83}$$

である. 仮定 5.6.1 および仮定 5.6.4 が実現可能であることが分かる.

つぎに, 仮定 5.6.3 および仮定 5.6.6 について説明する. これらをロボットのダイナミックス (5.82) において検証してみよう. (5.75) が成立していることはすぐに確認できるのであろう. (5.83) より,

$$\hat{h}(t, y_h(t)) = -(f(q, \dot{q}) + g(q)) \tag{5.84}$$

となることが分かる. つまり,

$$u_p(t) = -\hat{h}(t, y_h(t)) \tag{5.85}$$

と与えればロボットの非線形性を抑えることができる.

また, これについて $f(q, \dot{q}) + g(q)$ が有界であれば仮定 5.6.2 および仮定 5.6.7 が成立することになる. $f(q, \dot{q})$ および $g(q)$ の構成から, $g(q)$ が有界であることがわかる. また, 遠心力およびコリオリ力が有界であるので, 摩擦項が有界である限り, (5.77) 式が成立することになる.

最後に, 仮定 5.6.6 についてであるが, ロボット のダイナミックス (5.82) は ASPR 条件を満足しないため, フィードフォワード補償機構 (PFC = Parallel Feedforward Compensator) を付加する必要がある.

5.5.3 規範モデルの記述

この, 合成されたロボットのダイナミクス (5.12) に対して, 漸近安定な線形規範モデルを

$$\begin{aligned} \dot{x}_m(t) &= A_m x_m(t) + B_m u_m(t) \\ y_m(t) &= C_m x_m(t) \end{aligned} \tag{5.86}$$

とする. 実際, SAC では規範モデルを低次に選ぶことができるので, ここに

$$\begin{aligned} n_m &= m = n \\ q_m &= x_m \qquad\qquad q_m, x_m, u_m, y_m \in R^{m\times 1} \\ A_m &\in R^{m\times m} \end{aligned}$$

をとる.

5.5.4 フィードフォワード補償機構の記述

フィードフォワード補償特性を状態方程式の形で

$$\begin{aligned} \dot{x}_f(t) &= A_f x_f(t) + B_f u_p(t) \\ y_f(t) &= C_f x_f(t) \end{aligned} \tag{5.87}$$

ここで, フィードフォワード補償の元のプラントへの影響を最小限にするために B_f および C_f を小とする.

5.5.5 コントローラの設計

入力項

制御入力を

$$u_p(t) = u_a(t) = K(t)z(t) + u_R(t) \tag{5.88}$$

とする. 但し,

$$K(t) = [K_e(t), K_x(t), K_u(t)] \tag{5.89}$$

$$z(t) = [e_y(t)^T, x_m(t)^T, u_m(t)^T]^T \tag{5.90}$$

また, 非線形外乱抑制入力 $u_R(t) = [u_{R1}(t), \cdots, u_{Rm}(t)]^T$ は

$$\begin{aligned} H_i(t) &= [\rho_{0i},\ \rho_{1i}] \qquad \rho_{0i}, \rho_{1i} > 0 \\ z_h(t) &= \begin{bmatrix} 1 \\ ||y_h(t)|| \end{bmatrix} \\ e_y(t) &= \begin{bmatrix} e_{y_1}(t) \\ \vdots \\ e_{y_m}(t) \end{bmatrix} \end{aligned}$$

とするとき,

$$\begin{aligned} |u_{Ri}(t)| &\leq H_i(t)z_h(t) \\ u_{Ri}(t) &= \begin{cases} -H_i(t)z_h(t)\ \mathrm{sgn}\{e_{y_i}(t)\}, & |H_i(t)z_h(t)e_{y_i}(t)| > \epsilon \\ -\{H_i(t)z_h(t)\}^2 e_{y_i}(t)/\epsilon, & |H_i(t)z_h(t)e_{y_i}(t)| \leq \epsilon \end{cases} \end{aligned} \tag{5.91}$$

と与える. 但し, ϵ は微小正定数である.

$u_R(t)$ の計算では $y_h(t)$ が必要なので, ロボットの速度情報が必要になってくる. 速度情報を出力するエンコーダが付いていればそれを用いて $\dot{q}(t)$ を取ることができるが, 位置情報を出力するエンコーダだけしか付いていなければこの位置情報をコントローラのプログラム内で微分を取り, 速度情報を抽出する.

適応調整則

次のような積分比例型適応調整則を使う.

$$
\begin{aligned}
K(t) &= K_I(t) + K_P(t) && (5.92)\\
\dot{K}_I(t) &= -e_y(t)z(t)^T\Gamma_{K_I} - \sigma_K(t)K_I(t) && (5.93)\\
K_P(t) &= -e_y(t)z(t)^T\Gamma_{K_P} && (5.94)\\
H_i(t) &= H_{Ii}(t) + H_{Pi}(t) && (5.95)\\
\dot{H}_{Ii}(t) &= |e_{y_i}(t)|z_h(t)^T(\Gamma_{H_I})_i - \sigma_{Hi}(t)H_{Ii}(t) && (5.96)\\
H_{Pi}(t) &= |e_{y_i}(t)|z_h(t)^T(\Gamma_{H_P})_i \geq 0 && (5.97)
\end{aligned}
$$

$$
\begin{aligned}
\Gamma_{K_I} &= \Gamma_{K_I}^T > 0\\
\Gamma_{K_P} &= \Gamma_{K_P}^T > 0\\
(\Gamma_{H_I})_i &= (\Gamma_{H_I})_i^T > 0\\
(\Gamma_{H_P})_i &= (\Gamma_{H_P})_i^T > 0\\
\sigma_K(t) &= \sigma_{K0}\frac{e_y(t)^T e_y(t)}{1 + e_y(t)^T e_y(t)}\\
\sigma_{Hi}(t) &= \sigma_{H0i}\frac{e_{y_i}(t)^2}{1 + e_{y_i}(t)^2}\\
\sigma_{K0} &\quad , \quad \sigma_{H0i} > 0
\end{aligned}
$$

5.6 適応コントローラの簡略化

多関節ロボットのための多入出力単純適応制御コントローラの設計手順はとりあえずこれで完成だが, そのまま適用すれば上記の式からも分かるようにかなりの計算を必要としている.

今度は, 上記で述べたコントローラの設計手順に対して簡略化の方法を考えてみよう. ロボット (プラント) の伝達関数をさらに考慮すれば, このコントローラを簡略化することができる可能性がでてくる.

規範モデル

最小型比干渉系モデルとする.

5.6.1 線形部の簡略化

さて, ロボットダイナミックスの非線形項 $h(t, y_h(t))$ が完全に補償されたとき, (5.10) 式よりロボットの伝達関数行列が

$$G_p = \frac{1}{s^2} I_m Q \tag{5.98}$$

であることが分かり, これは極めて特殊な形をしており, 伝達関数行列は慣性行列 Q だけに依存していることが分かる. この, 慣性行列の形さえ把握できれば, 適応コントローラの簡略化が考えられる.

実際, J_0 の定義 ((5.79) 式) より, J_0 が対角行列の形をしていることが分かる. また, 慣性行列 $R(q)$ は帯行列の形をしているが, これは, 仮定 5.6.8 より J_0 に比較的に小である. よって, 慣性行列 Q^{-1} が対角行列に近似することができる.

以上のことから, 次のことが言える.

第 4 章に習って, 次のように適応フィードバック係数の計算を簡略化する.

定理 5.1

適応コントローラの簡略化

- ${Q^{-1}}_{ij} \approx 0$ であるとき,

$$\begin{aligned} K_{eij} &= 0 \\ K_{xij} &= 0 \\ K_{uij} &= 0 \end{aligned} \tag{5.99}$$

また, ほとんどのロボットでは, アームは直列に接続されている場合が多いことが分かる. これより, 慣性行列 Q^{-1} が対角行列または帯状の形式をとっていることを推測できる.

5.6.2 非線形部の簡略化

非線形補償機構については, 妥当な簡略化が考えにくいが, 計算をなるべく少なくするという目的であれば, 次のことが考えられる.

1. (5.93) および (5.96) において σ を時変のかわりに正定数を用いる.

2. 次の仮定をおく.

仮定 5.7

(5.77) 式で述べた非線形項の有界条件を次のように変更する.

$$|\hat{h}_i(t, y_h(t))| \leq \rho_{0i} \tag{5.100}$$

この仮定の下, (5.91) 式にかわって非線形外乱抑制入力 $u_R(t) = [u_{R1}(t), \cdots, u_{Rm}(t)]^T$ は

$$\begin{aligned} H_i(t) &= \rho_{0i} \geq 0 \\ e_y(t) &= \begin{bmatrix} e_{y_1}(t) \\ \vdots \\ e_{y_m}(t) \end{bmatrix} \end{aligned}$$

とするとき,

$$\begin{aligned} |u_{Ri}(t)| &\leq H_i(t) \\ u_{Ri}(t) &= \begin{cases} -H_i(t)\ \mathrm{sgn}\{e_{y_i}(t)\}, & |H_i(t) e_{y_i}(t)| > \epsilon \\ -H_i(t)^2 e_{y_i}(t)/\epsilon, & |H_i(t) e_{y_i}(t)| \leq \epsilon \end{cases} \end{aligned} \tag{5.101}$$

と与える. 但し, ϵ は微小正定数である.

また, 次の仮定が成立すれば適応コントローラがさらにスマートとなる.

仮定 5.8

$$|k_i(t, y(t))| + |\hat{h}_i(t, y_h(t))| \leq \rho_{0i} \tag{5.102}$$

但し $k_i(t, y)$ は i 番目のサブシステムに対する干渉である.

以上のことが成立すれば, (5.89) 式の代わりに

$$K(t) = [\mathrm{diag}[k_{ei}(t)]_{i=1}^{m}, \mathrm{diag}[k_{xi}(t)]_{i=1}^{m}, \mathrm{diag}[k_{ui}(t)]_{i=1}^{m},] \tag{5.103}$$

を使う.

ただ, この場合, コントローラが完全に分散型となるが, 干渉項の補償も非線形補償機構に任せることになるので, 非線形補償機構の負担は更に重くなる.

5.6.3 超多自由度ロボット制御への挑戦

以上の手順に従い, 多関節ロボットのためのコントローラを設計する. 更に, 入力および出力の数を増やし, 超多自由度ロボットのアーム制御に適用してみる.

超多自由度ロボットとは, 関節の数が極端に多い, 蛇のようなロボットのことを言う.

そのシミュレーション結果を次章にて詳しく記載されているが, ここでお断りしたいのは, 超多自由度ロボットでは, 関節の数が極端に多いことで, その関節間に干渉が発生しやすい. 特に, 根もとに近い関節では, 他の全ての関節から力が加わり, 制御がしにくいのである. ところがこれも, 時間がたてば, 補償されるようになることが, シミュレーションの結果からも実証される.

6 ロボット適応制御シミュレーション

この章では, 第 4 章および第 5 章で述べた方法を用いて Matlab や機構解析プログラム DADS によるプログラムを作成し, そのシミュレーション結果を記載する.

6.1 SISO-SAC によるロボットの単関節制御

6.1.1 モデル

単変数の単純適応制御によるロボットのアーム制御シミュレーションにおけるコントローラの設計を図 6.3 および図 6.2 に示す. モデルとして使用したロボットは図 6.1 である. 3 関節 (3 自由度) ロボットではあるが, ここでは第 1 関節および第 3 関節の入力および出力を無効にしている. つまり, 第 2 関節だけを制御している.

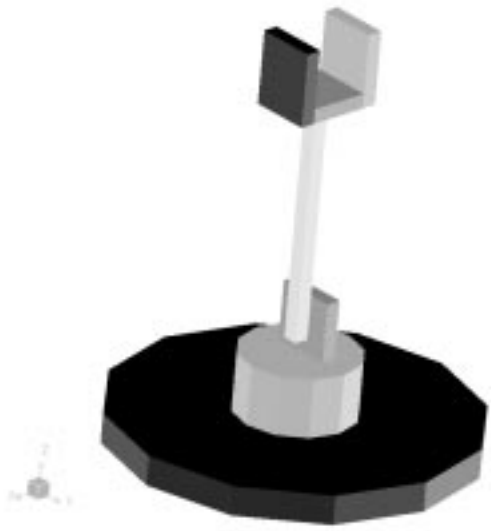


図 6.1: 3 関節ロボットモデル

6.1.2 制御シミュレーション結果と考察

制御結果グラフは図 6.4 に示す. 干渉や外乱がうまく補償されていることが伺える.

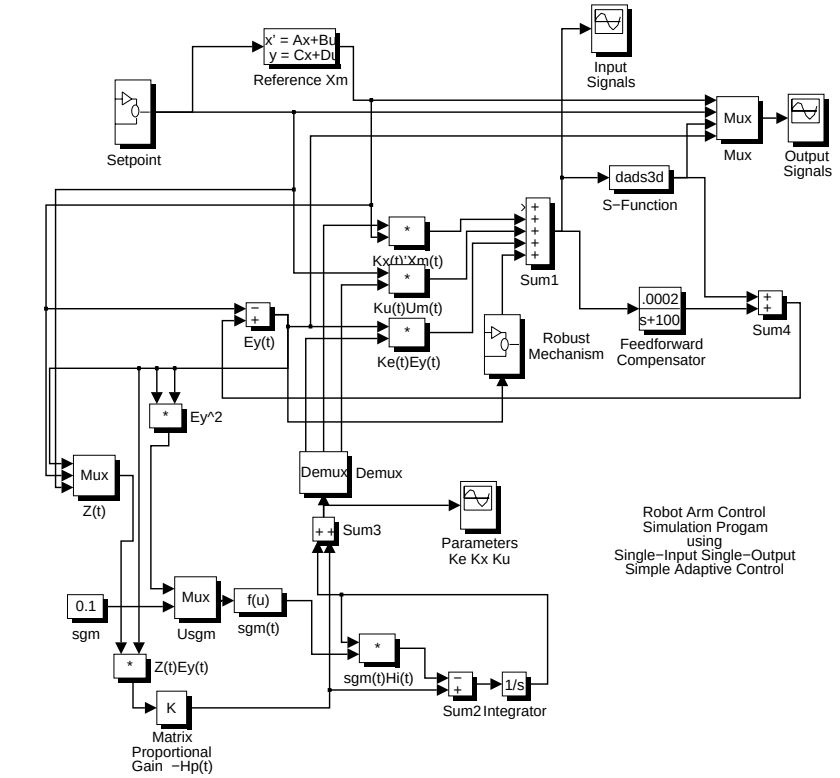


図 6.2: SISO-SAC によるロボットの単関節制御コントローラ

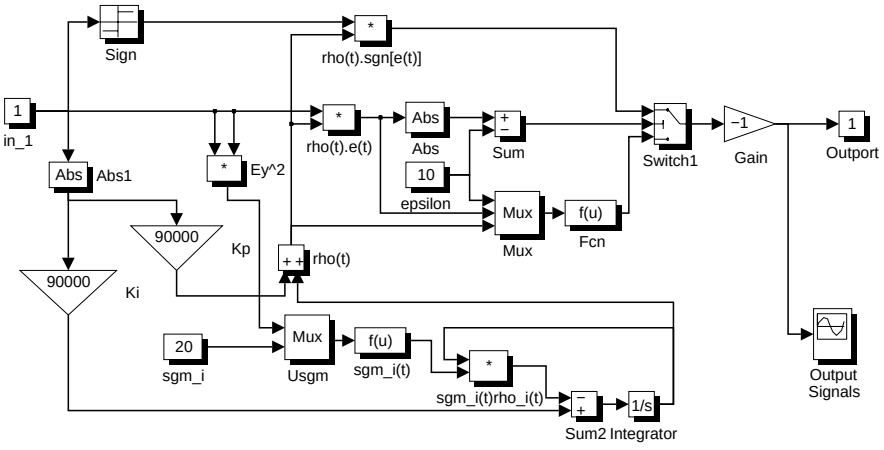
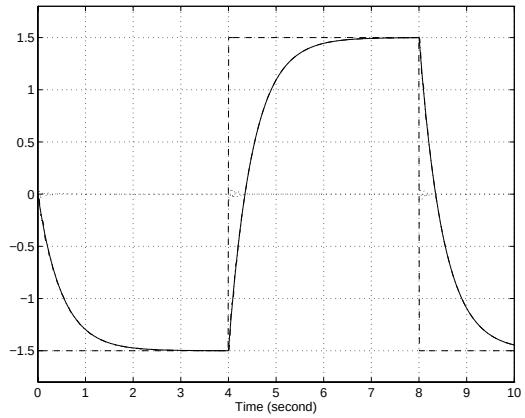
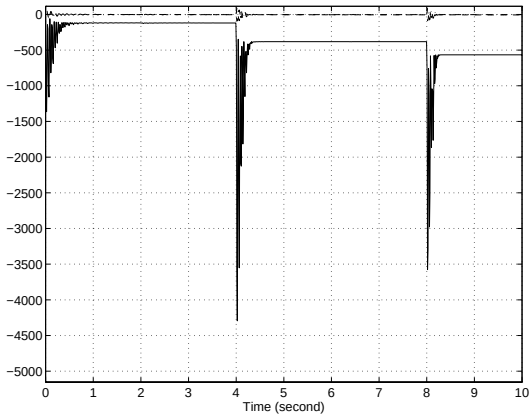


図 6.3: SISO-SAC によるロボットの単関節制御 (ロバスト機構)

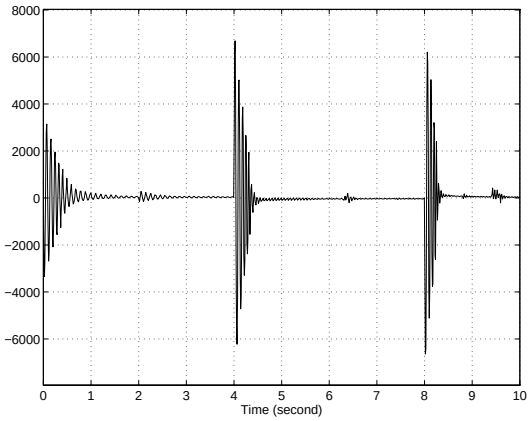
図 6.4 より, 制御対象 (プラント) の出力と規範モデルの出力との誤差 (出力追従誤差) が, 初期運動を除いて, 時間がたつにつれてゼロに向かっていくことが分かる. プラントのパラメータを適応同定則によって適応的に推定し, 時間の経過とともに推定パラメータの値は正しい値に収束している.



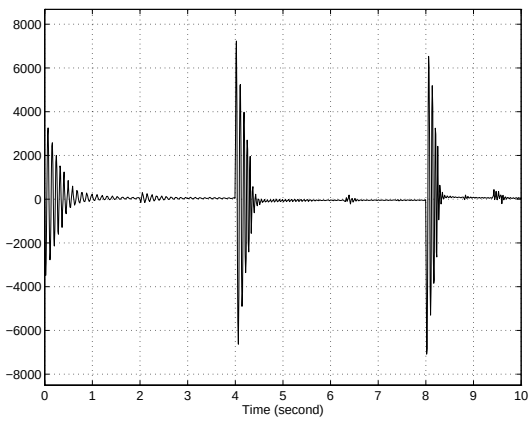
(a) プラント出力, 追従誤差, モデル入力



(b) 適応パラメータ K_e, K_x, K_u



(c) ロバスト機構出力



(d) プラント入力

図 6.4: SISO-SAC によるロボット制御シミュレーション結果

その動きは図 6.5 に示す. このシミュレーションでは, DADS で作成したロボットのモデルを Matlab 形式の特殊な S-Function ファイルとして落し, それを Simulink 形式のシミュレーションプログラムの中に取り入れて制御対象として扱う.

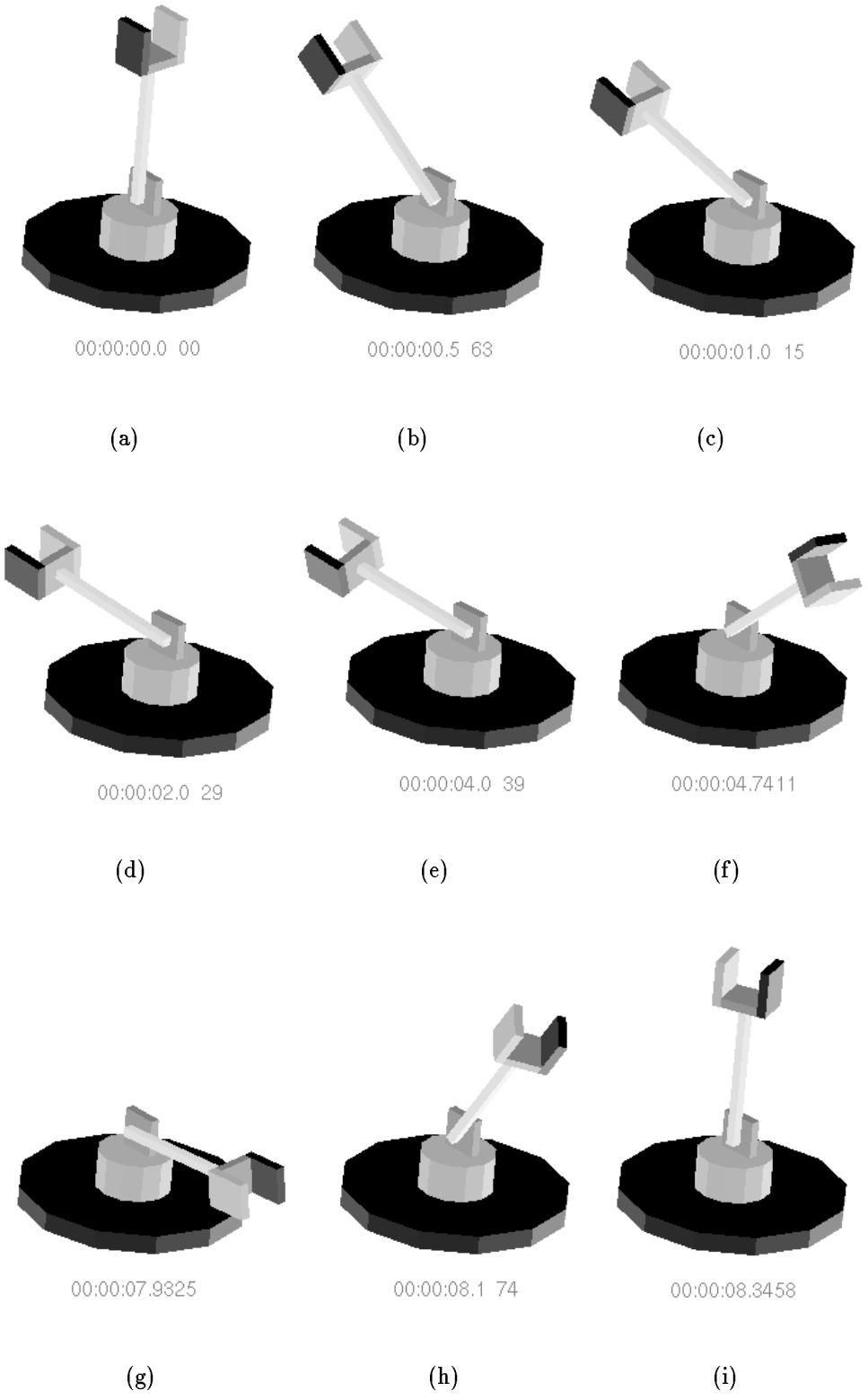


図 6.5: SISO-SAC によるロボット制御シミュレーション結果

6.2 MIMO-SAC によるロボット制御シミュレーション

次は MIMO-SAC によるロボット制御シミュレーションの結果を示す.

6.2.1 モデル

多入出力の単純適応制御によるロボットのアーム制御シミュレーションにおけるコントローラの設計を図 6.7 に示す. モデルとして使用したロボットは図 6.6 である. 5 関節 (5 自由度) ロボットではあるが, ここでは第 4 関節および第 5 関節の入力および出力を無効にしている. つまり, 根元から第 1 関節 ～ 第 3 関節だけを制御している.

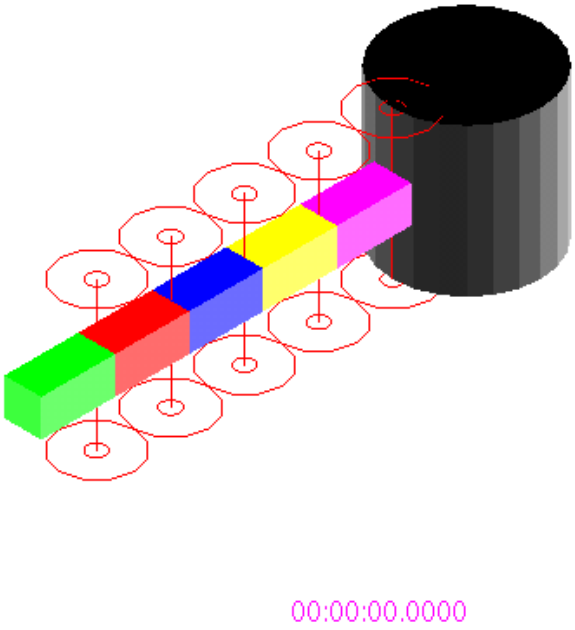


図 6.6: 5 関節ロボットモデル

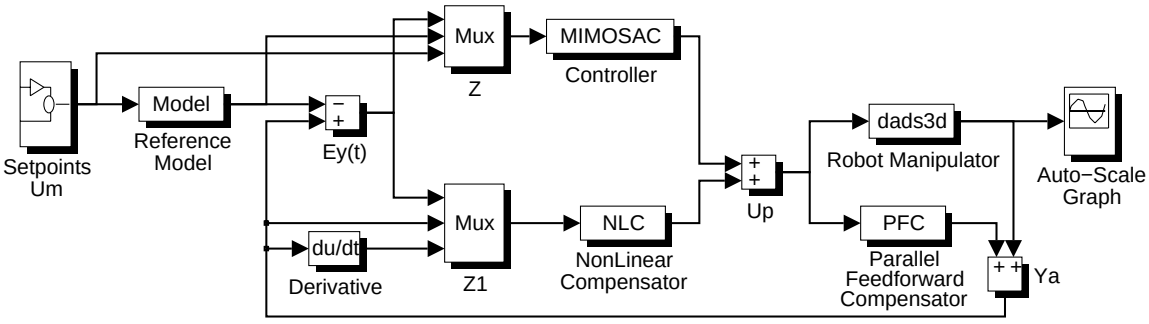


図 6.7: MIMO-SAC によるロボットの関節制御コントローラ

このシミュレーションでは, C-MEX による S-Function ブロックが使われており, そのプログラムソースは付録 A に示す (Model.c, PFC.c, MIMOSAC.c, NLC.c).

6.2.2 制御シミュレーション結果と考察

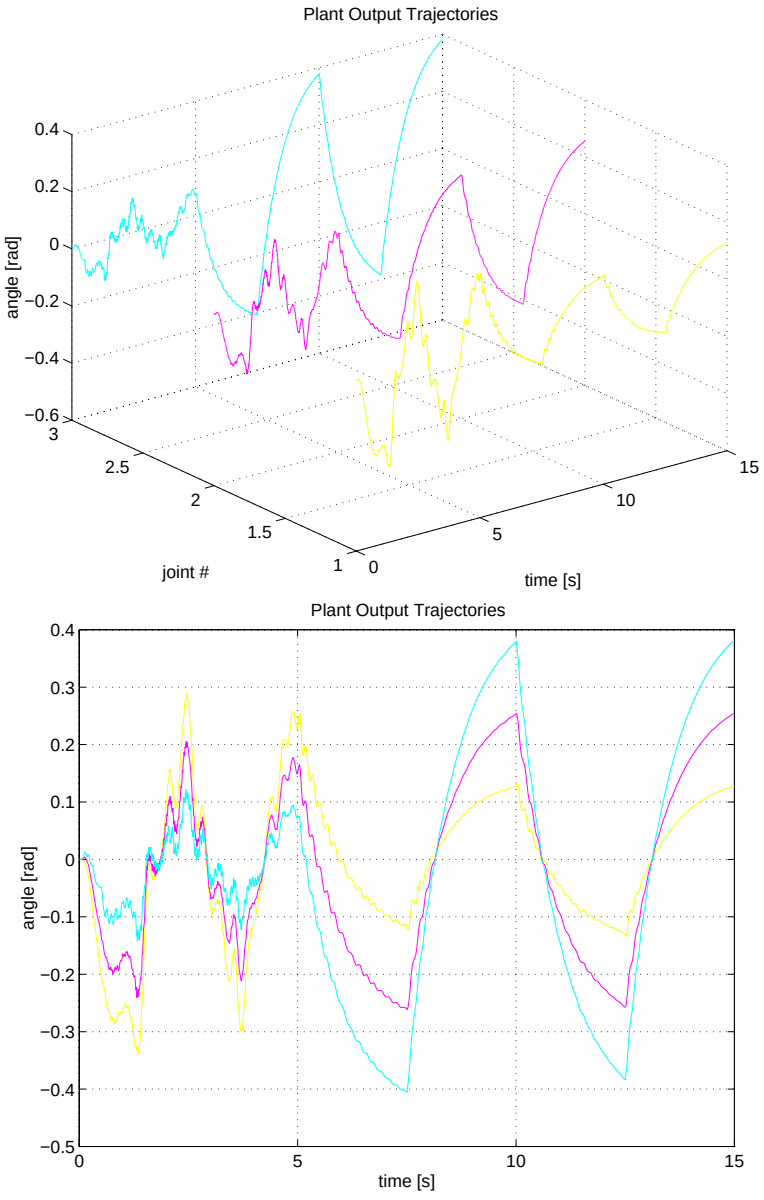


図 6.8: MIMOSAC によるロボット制御シミュレーション結果 (各関節の相対角度出力)

制御結果グラフは図 6.8 に示す. これはプラント (ロボット) の出力 (第 1 関節 〜 第 3 関節のそれぞれの相対角度) だけを示すグラフであるが, 関節間干渉がだんだん補償されていくことが伺える.

但し, ここでは規範モデルの入力として矩形波信号が与えられており, $t = 5$ s の時入力信号の波形および周期の切替えが生じているものとする.

適応パラメータの調整器において, その調整ゲインを高くして調節すれば, 図 6.9 のように, 収束性を高めることが可能である.

ここでは, コントローラの構成として従来法の多入力多出力単純適応制御法に基づくものを用いている. 変数の数 (入力および出力の数) が少ないこのシミュレーションでは従来のものでも十分に適用が可能であるが, 入出力の数がさらに増加すれば従来法の多入力多出力単純適応制御法に基づくコントローラの実現が困難になってくる.

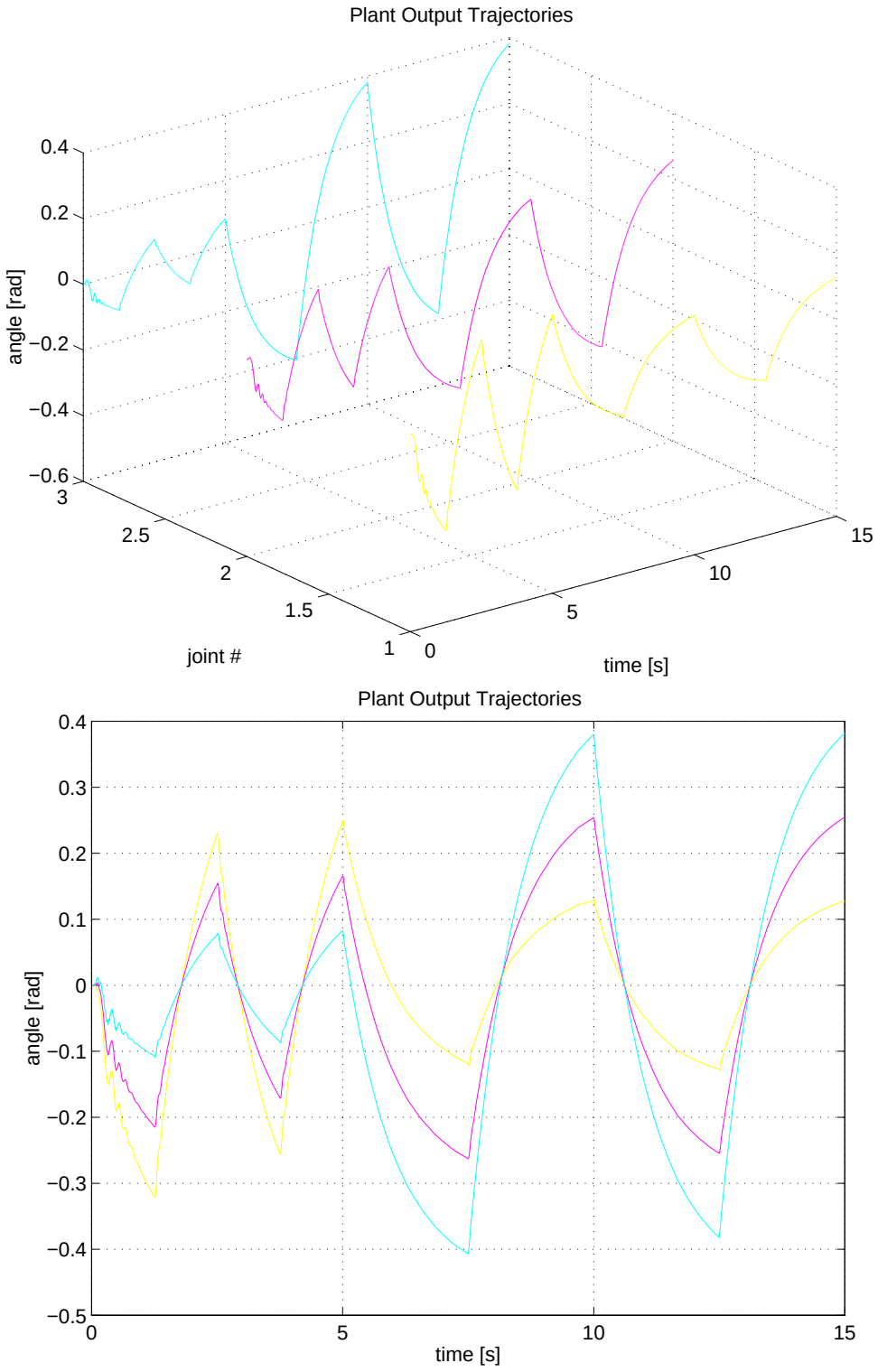


図 6.9: MIMOSAC によるロボット制御シミュレーション結果 (各関節の相対角度出力)

6.3 簡略型 SAC による多関節ロボット制御

次は簡略型 SAC によるロボット制御シミュレーションの結果を示す.

6.3.1 モデル

多入出力の単純適応制御によるロボットのアーム制御シミュレーションにおけるコントローラの設計を図 6.10 に示す. 前回のシミュレーションとの比較のため, モデルとして使用したロボットは前回のシミュレーションと同様なもの (図 6.6) にしている. 5 関節 (5 自由度) ロボットではあるが, ここでは第 4 関節および第 5 関節の入力および出力を無効にしている. つまり, 根元から第 1 関節 〜 第 3 関節だけを制御しているものとする.

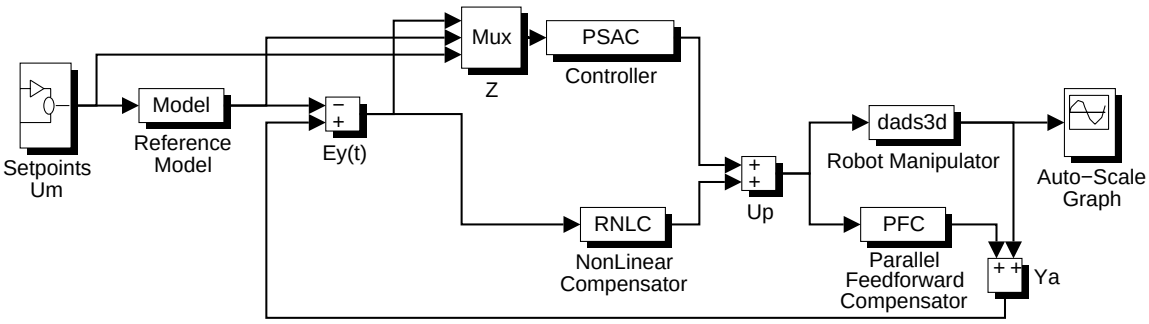


図 6.10: 簡略型 MIMO-SAC によるロボットの関節制御コントローラ

このシミュレーションでは, C-MEX による S-Function ブロックが使われており, そのプログラムソースは付録 A に示す (Model.c,PFC.c,PSAC.c,RNLC.c).

図 6.10 は一見, 普通の MIMO-SAC の構成図 (図 6.7) に見えるかも知れないが, 図 6.7 に比べて, 図 6.10 における適応コントローラ (図 6.10 では PSAC と記されている) にはかなりの簡略化がなされている. そのソースファイル (付録 A の PSAC.c 参照) よりも分かるように, コントローラを分散的に配置しているので, 構造的には一番単純な形をしており, 使用する計算機は少ないので計算量も当然軽減される.

また, 非線形性補償機構 (図 6.7 では NLC, 図 6.10 では RNLC) については, 微分器を使用せず, 微分情報を考慮しないことにし, その計算量を減らす事で簡略化を図る.

6.3.2 制御シミュレーション結果と考察

制御結果グラフは図 6.11 に示す. これはプラント (ロボット) の出力 (第 1 関節 ～ 第 3 関節のそれぞれの相対角度) だけを示すグラフであるが, 関節間干渉がだんだん補償されていくことが伺える.

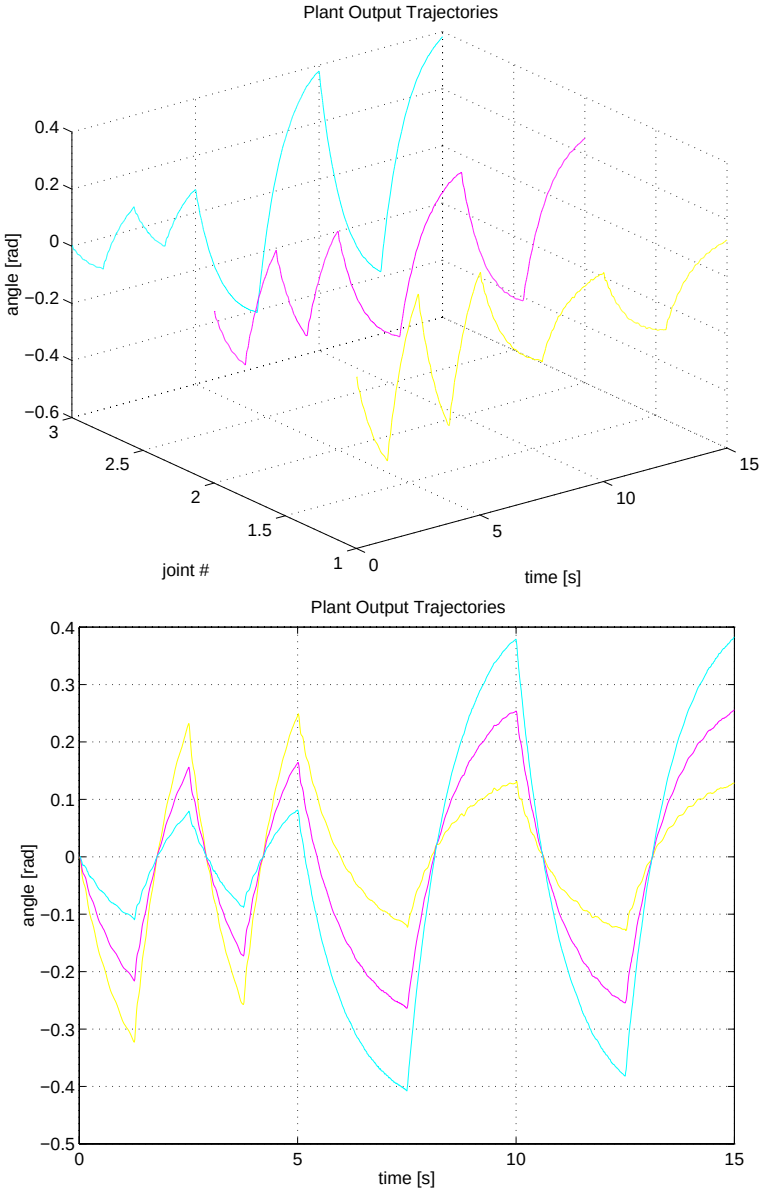


図 6.11: 簡略型 MIMO-SAC によるロボット制御シミュレーション結果 (各関節の相対角度出力)

但し, ここでは規範モデルの入力として矩形波信号が与えられており, $t = 5$ s の時入力信号の波形および周期の切替えが生じているものとする.

適応パラメータの調整器において, その調整ゲインを高くして調節すれば, 図 6.12 のように, 収束性を高めることが可能である.

ここでは, コントローラの構成として従来法の多入力多出力単純適応制御法に基づくものに対して簡略化を行い, その簡略型コントローラを用いている. この方法は, 変数の数 (入力および出力の数) が多い場合には特にその威力を発揮する. 入出力の数が増えるとコントローラ器の数が n^2 のオーダーで増加する標準の多入出力単純適応制御手法に対して, この方法ではたった

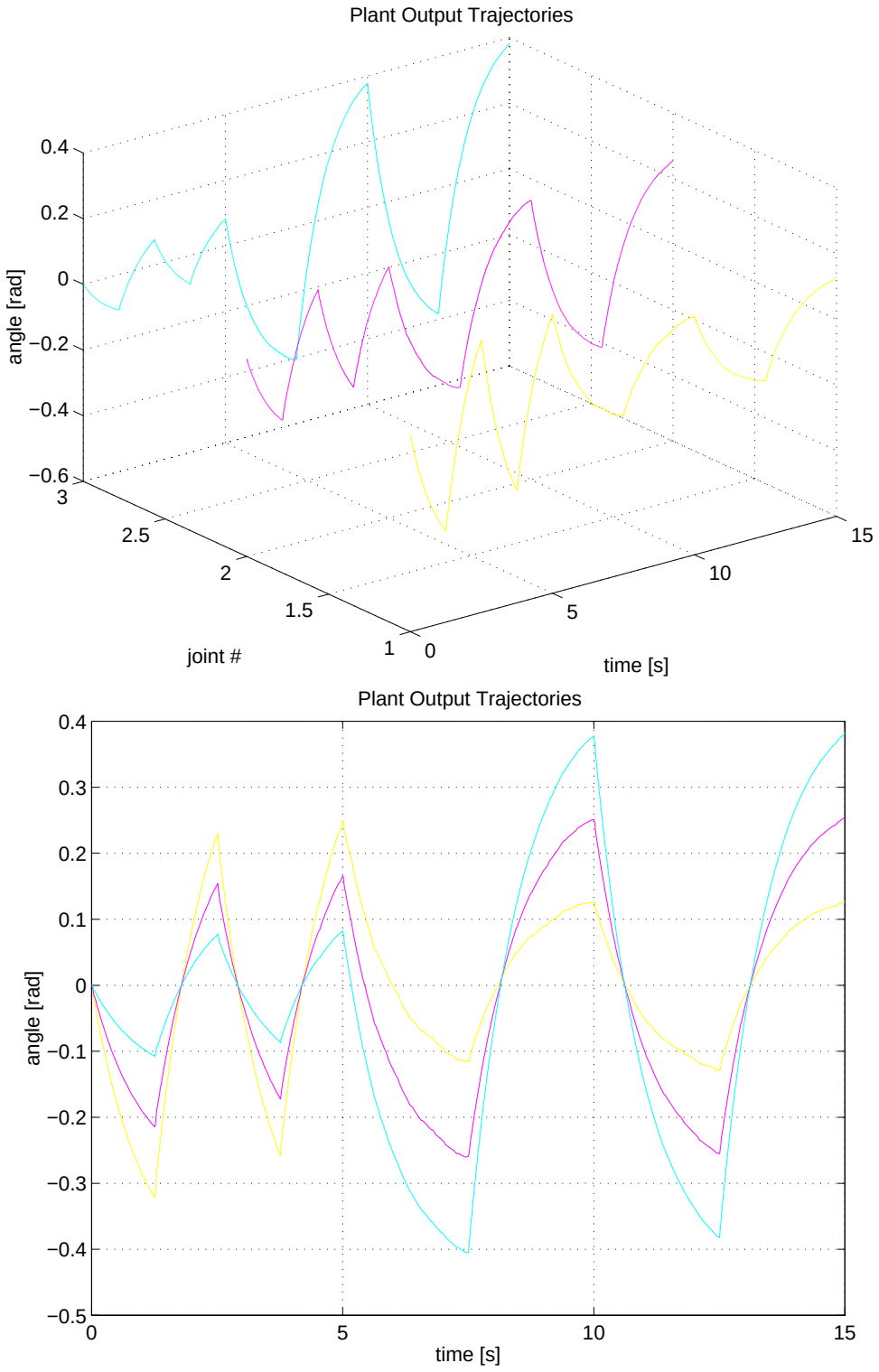


図 6.12: MIMOSAC によるロボット制御シミュレーション結果 (各関節の相対角度出力)

の n のオーダーでコントローラを構成できる.

6.4 超多自由度ロボット

多入出力系の簡略型単純適応制御による超多自由度ロボット (**超多関節ロボット**) のアーム制御シミュレーションにおけるコントローラは図 6.10 に示したものを利用する.

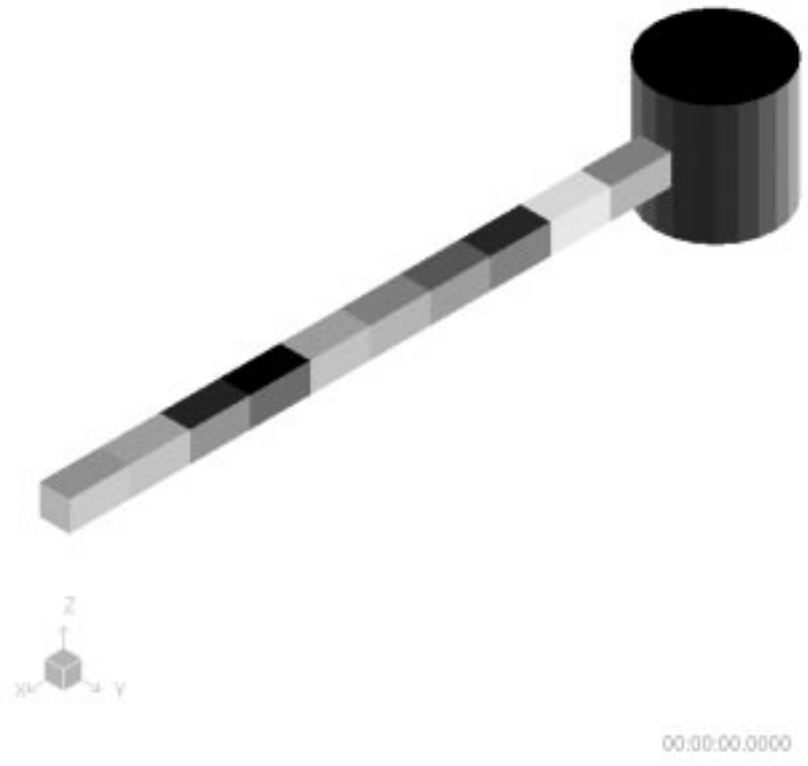


図 6.13: 超多関節ロボットモデル (10 関節)

モデルは図 6.13 とし, 10 個の関節をもつ蛇型ロボットマニピュレータとする.

制御結果グラフは図 6.14 に示す. この, 超多自由度ロボットの制御は, 標準の MIMOSAC を用いるとコントローラの数 (積分器など) が, 適応調整則を実行するだけのために, およそ 300 個が必要である. これは, かなりの予算が必要ということになり, 事実上, 実現し兼ねる.

提案した簡略型 SAC コントローラを用いると, 最低限 30 個のコントローラ成分画あれば十分なので, 実現はしやすい.

この図からも分かるように, 根元に近い関節は負荷が大きいため干渉や非線形性の影響を受けやすいが, 時間がたつに連れてだんだん応答特性が良くなる. 干渉や非線形性は, 特に根元に近い関節は, ゆっくりではあるが, 時間がたつに連れて補償されるようになることが伺える.

なお, このロボットの動きは図 6.15 〜 図 6.17 に示す.

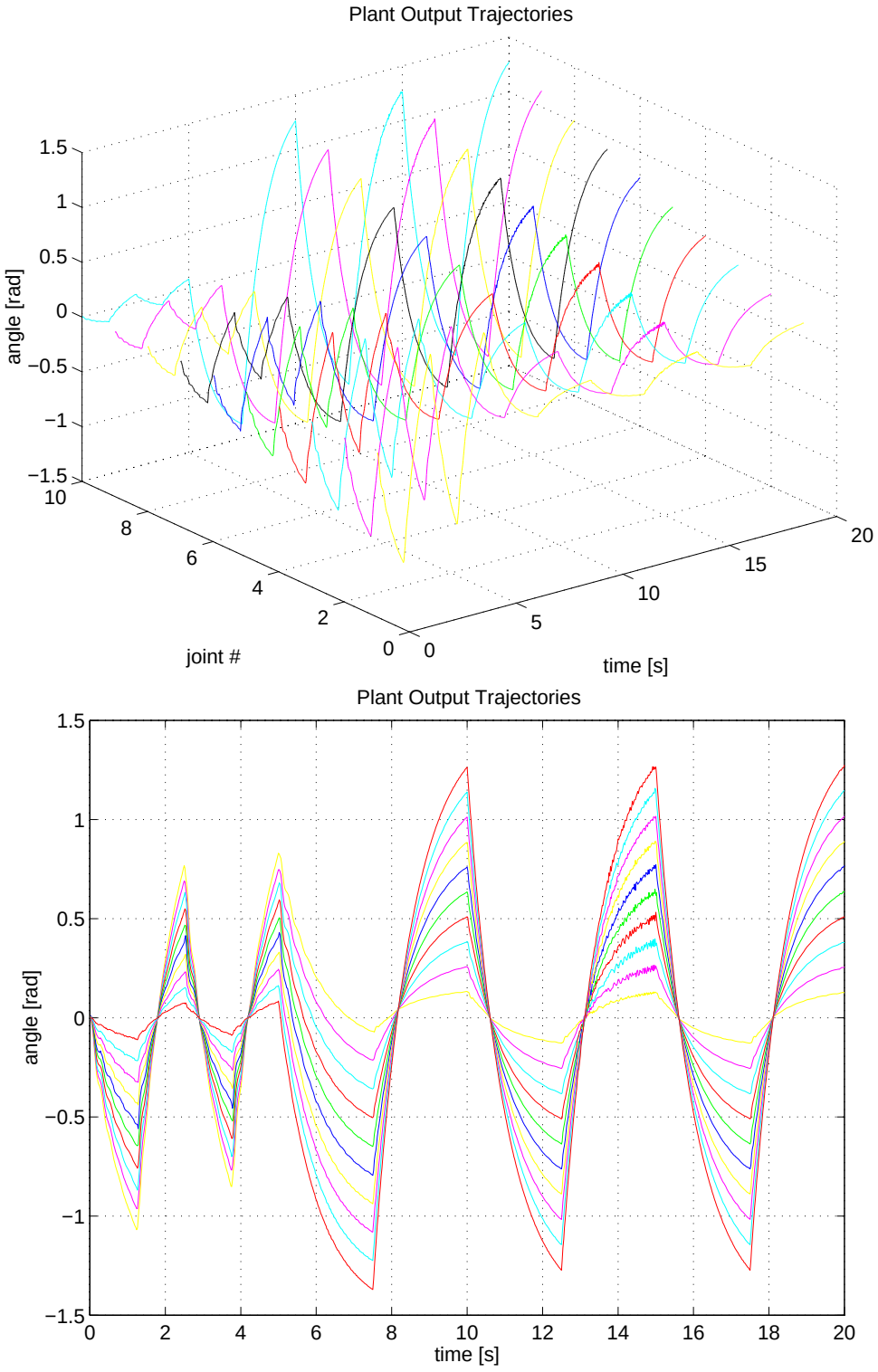


図 6.14: 簡略型 SAC による超多自由度ロボット制御シミュレーション結果

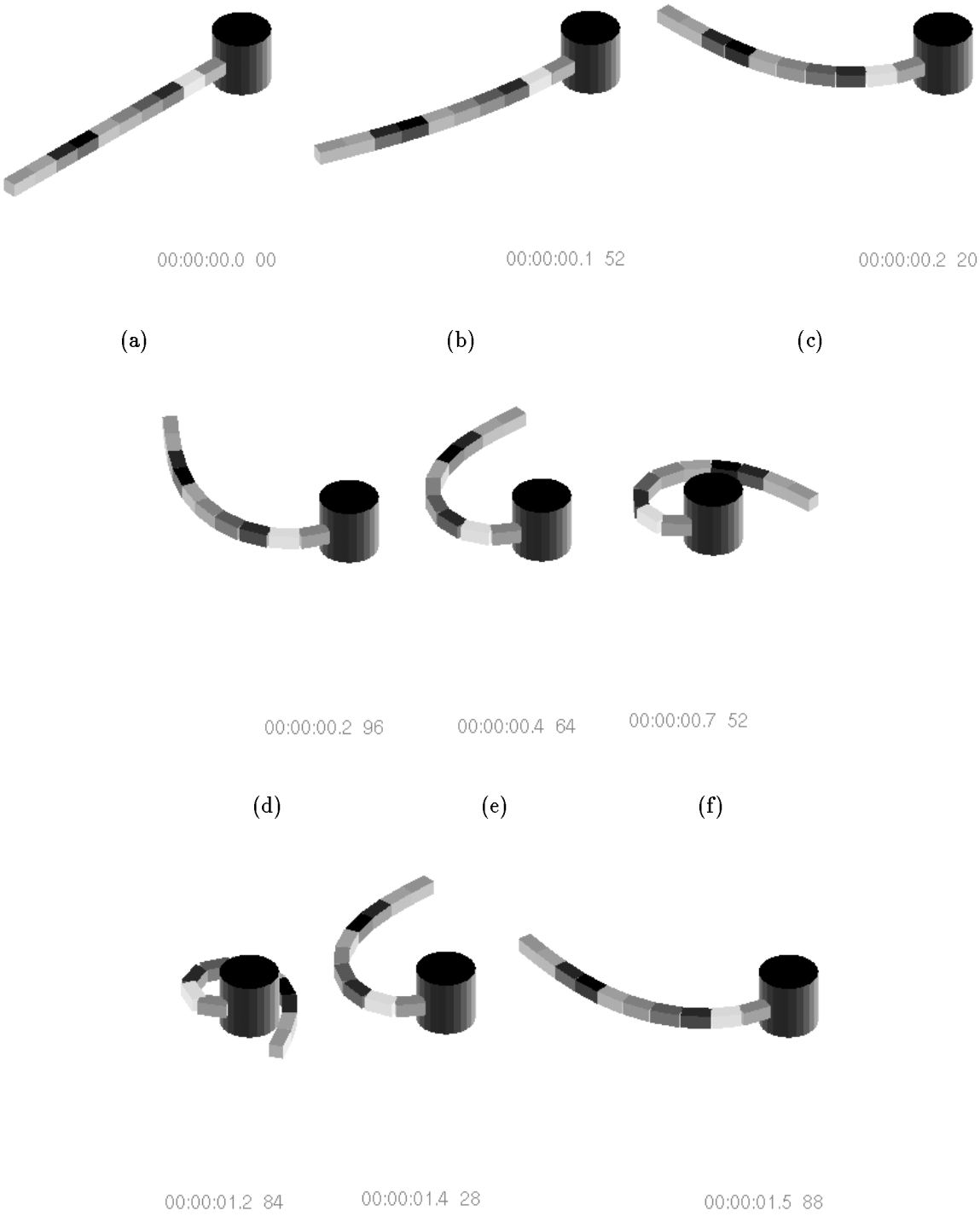


図 6.15: 簡略型 SAC による超多自由度ロボット制御シミュレーション結果 (その 1)

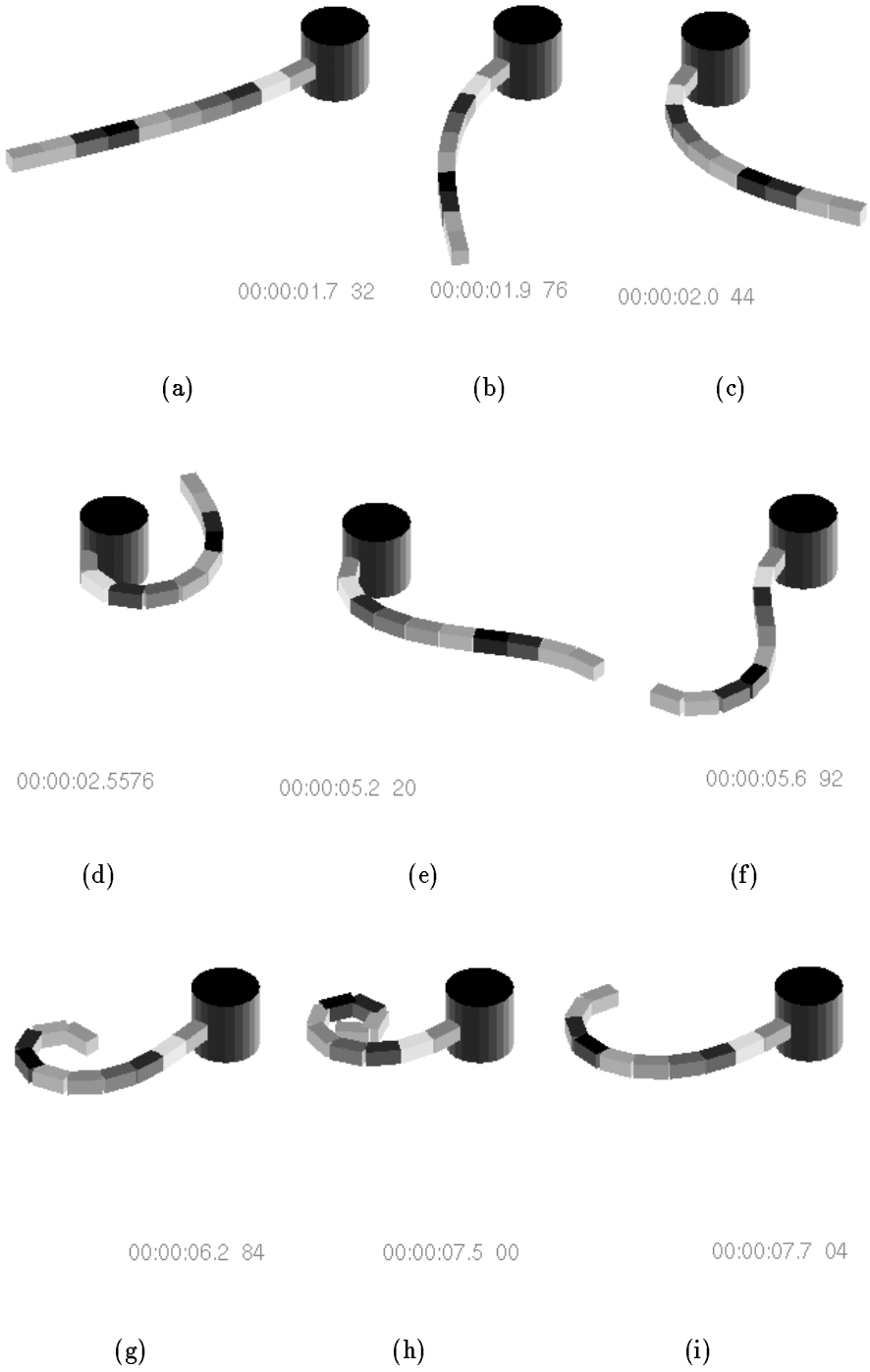


図 6.16: 簡略型 SAC による超多自由度ロボット制御シミュレーション結果 (その 2)

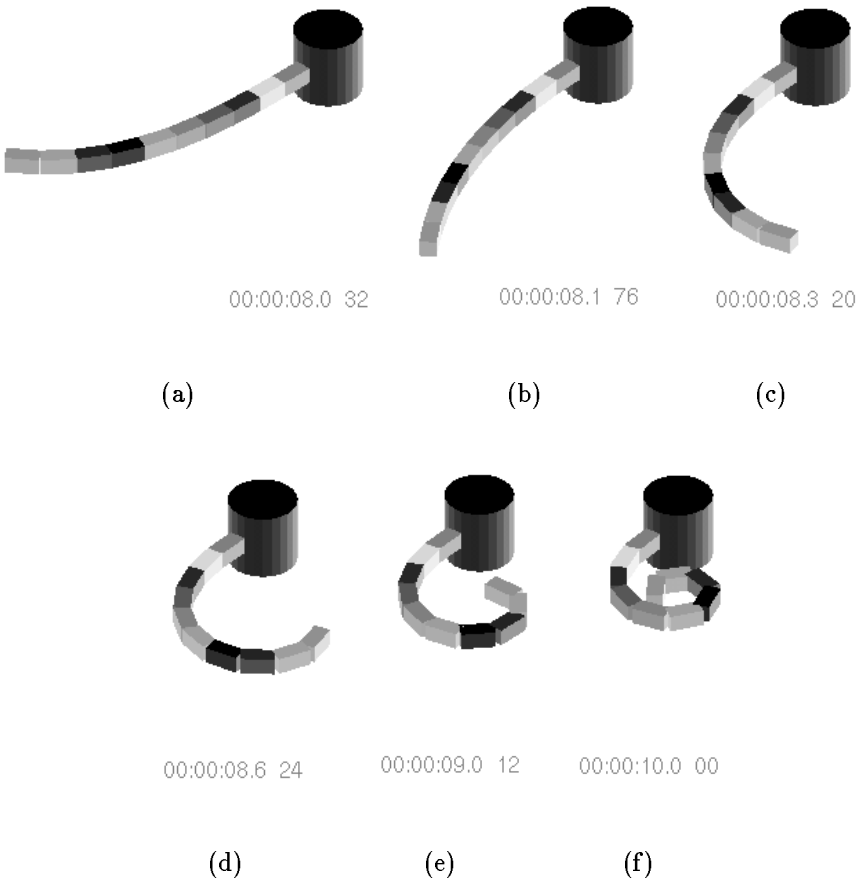


図 6.17: 簡略型 SAC による超多自由度ロボット制御シミュレーション結果 (その 3)

7 結言

本研究では, 多入力多出力系のための多変数単純適応制御における計算量の問題を明確にし, それを克服するための対策を提案した. 考案したコントローラの簡略化手法をシミュレーションによって検討し, その結果の性能評価を行った.

また, 単純適応制御手法をロボットマニピュレータのアーム制御に適用した時の起こり得る軸間干渉や非線形性など, それらの問題点を明確にし, それを克服するための方法を考えた.

さらに, 考案した簡略型制御手法を用いて, 超多関節ロボットの超多自由度アーム制御に挑戦を試み, 機構解析プログラム DADS によるシミュレーションの結果を検討した. 標準の MIMO-SAC では実現がほぼ不可能な 10 関節ロボットをモデルとして使ったこのシミュレーションは, 案した簡略型制御ではなんとか実現可能になった. 時間がたつにつれてロボットの動きが良くなることもシミュレーションの結果より確認できた.

標準の MIMO-SAC に比べては言うまでもなく良好な結果が得られたが, 完全とは言えず, 良好な追従特性が得られるまではかなりの時間がかかった. 理論上ではゲインをいくら高くしても良いのだが, 実世界ではやはり限界があり, ゲインが高すぎると計算が狂ってしまうことがある. これを次の課題にしたい.

最後になりましたが, 示村 悦二郎 教授, 藤田 政之 助教授, 増淵 泉 助手, 堀口 進 教授, 並びに示村・藤田ロボティクス研究室の皆様に, 本研究にあたり多大なる御指導を頂き, 心より感謝致します.

参考文献

[1] K. Sobel, H. Kaufman and L. Mabius : Implicit Adaptive Control for a Class of MIMO Systems, IEEE Trans. Aerospace and Electronic Systems, Vol. AES-18, No. 5, 576/589 (1982)

[2] 岩井 善太 : 構造の簡単な適応制御 (SAC), コンピュートロール No. 32, 66/72 (1990)

[3] 岩井, 水本 : 多入出力系の単純適応制御—並列フィードフォワード補償に基づく一般設計, 計測自動制御学会論文集, Vol. 29, No. 2, 159/168 (1993)

[4] 岩井, 大友, 水本 : 構造の簡単なロバスト適応制御系, 計測自動制御学会論文集, Vol. 27, No. 3, 306/313 (1991)

[5] 岩井, 水本, 足立, 堂園 : 単純適応制御手法による分散適応制御, 計測自動制御学会論文集, 28-9, 1052/1060 (1992)

[6] I. Bar-Kana and H. Kaufman : Simple Adaptive Control of Uncertain Systems, Int. J. Adaptive Control and Signal Processing, Vol. 2, 133/143 (1988)

[7] I. Bar-Kana and H. Kaufman : Robust Simplified Adaptive Control for a Class of Multivariable Continuous-Time Systems, 24th IEEE CDC 141/146 (1985)

[8] I. Bar-Kana and H. Kaufman : Low-Order Model Reference Direct Multivariable Adaptive Control, Proceedings of ACC 1259/1264 (1984)

[9] I. Bar-Kana : Parallel Feedforward and Simplified Adaptive Control, Int. J. Adaptive Control and Signal Processing, Vol. 1, No. 2, 95/109 (1987)

[10] J. R. Broussard and M. J. O' Brien : Feedforward Control to Track the Output of a Forced Model, IEEE Trans., Vol. AC-25, No. 4, 851/853 (1980)

[11] G. Strang 著, 山口 昌哉, 井上 昭 訳 : **線形代数とその応用**, 産業図書 (1993)

[12] 有本 卓 著 : **ロボットの力学と制御**, 朝倉書店 (1990)

[13] Karl J. Astrom and Bjorn Wittenmark : Adaptive Control (second edition), Addison-Wesley (1995)

[14] P.A. Ioannou : Robust Adaptive Control, Prentice Hall Inc. (1996)

A プログラム ソース

msacsim2.c

```
/*
 */
/*
┌──────────────────────────────────────┐
│                                      │
│    Program : msacsim2.c              │
│    Desc.   : MIMO-SAC Simulation Program
│    Creator : Budi Rachmanto 1996-11-14
│                                      │
│    多変数単純適応制御による制御シミュレーション    │
│                                      │
└──────────────────────────────────────┘
Plant :
  .      [ -1 1  0  0 ]       [1 0 0 0]
  Xp(t) = [-2 -1  1  0 ] Xp(t) + [0 1 0 0] Up(t)
         [ 0 -2 -1  1 ]        [0 0 1 0]
         [ 0  0 -2 -1 ]        [0 0 0 1]
         [ 1  0  0  0 ]
  Yp     = [ 0  1  0  0 ] Xp(t)
         [ 0  0  1  0 ]
         [ 0  0  0  1 ]
Criterion :
  Am  = -I*4
  Bm  = Cm = I*4
  Gm  = diag[1/(s+1), 1/(s+1), 1/(s+1), 1/(s+1)]
Parameters :
  GammaI = diag[10^8I'2, 10^3I'4]
  GammaP = diag[10^6I'2, 10^2I'4]
*/
#include ¡stdio.h¿
#include ¡math.h¿
#define PLANT_MODEL  0
#define CRITERION    1
#define ADJUSTMENT   2
#define YES          1
#define NO           0
#define PI           3.141592654
#define SamplingTime 0.01              /* サンプリング時間 */
#define MAX_COUNT    4000              /* 最大カウント数 */
#define period       20.               /* 入力の周期 */
#define Np           4                 /* プラント次数 */
#define Nm           4                 /* 規範モデル次数 */
#define R            4                 /* 入力次数 */
#define M            4                 /* 出力次数 */
#define amplitude    1.0               /* 矩形波の入力値 */
#define step         0.0               /* ステップ入力 */
#define epsilon      .0000001
#define Sgm1         0.01              /* 設計パラメータ */
#define Sgm2         0.05
/* Macros */
#define Ksize        (M+Nm+R)
#define notzero(x)   (fabs(x)¿epsilon) ? x : 0.
static double Um[R],Up[R],             /* 入力関数 */
      Uu[R],Ux[Nm],Ue[M],
```

```
      TIME = 0.0,                          /* 経過時間 */
      GammaI[Ksize]                         /* 本当は Gamma[Ksize][Ksize]*/
        =-100.,100.,100.,100.,10.,10.,10.,10.,10.,10.,10.,10.",
      GammaP[Ksize]
        =-100.,100.,100.,100.,10.,10.,10.,10.,10.,10.,10.,10.",
      Ki[R][Ksize],Kp[R][Ksize],           /* 調整則行列 */
      K[R][Ksize],Z[Ksize],
      Ap[Np][Np]=—1,0,0,0, 0,-1,0,0, 0,0,-1,0, 0,0,0,-1",
      Xp[Np]=-.0, .0, .0, .0",
      Xm[Nm]=-.0, .0, .0, .0",
      Ey[M]=-.0,.0,.0,.0",Ya[M]=-.0,.0,.0,.0",Yp[M],Ym[M],
      Ke[M][M] =-.0,.0,.0,.0",             /* 調整パラメータ */
      Kx[M][Nm]=-.0,.0,.0,.0",              /* (Tuning Parameters) */
      Ku[M][R] =-.0,.0,.0,.0";
/*
┌──────────────────────────────┐
│  = Criterion`Path Generator =  │
│                  |           │
│  ⊔⊓⊔⊓⊔⊓⊔⊓⊔⊓⊔  │
│                  |           │
│  規範モデル入力を決定する関数  │
│  周期 period で矩形波を生成    │
└──────────────────────────────┘
*/
double StepRectangle(double t,double u)
{
   return( (fmod(t,period) ¡= period/2) ? u+step : -u+step );
}

/*
┌──────────────────────────────┐
│  状態方程式を用いて次の状態を出力  │
└──────────────────────────────┘
*/
NextState(X,Xdot,mode)
   int mode;
   double *X, *Xdot;
{
   int i,j;
   double SigmaI,EQuad;
   switch(mode){
     case PLANT`MODEL :
       for(i=0;i¡Np;i++){
         for(j=0,Xdot[i]=0;j¡Np;j++) Xdot[i]+=Ap[i][j]*X[j];
  Xdot[i]+=Up[i];
  Yp[i]=X[i];
       }
       break;
     case CRITERION :
       for(i=0;i¡Nm;i++) Xdot[i] = -(Ym[i]=X[i]) + Um[i];
       break;
     case ADJUSTMENT :
for(i=0,EQuad=0.;i¡M;i++) EQuad += Ey[i]*Ey[i];
       SigmaI = Sgm1 * EQuad / (1.0 + EQuad) + Sgm2;
for(i=0;i¡R;i++) for(j=0;j¡Ksize;j++)
        *(Xdot+Ksize*i+j) = -Ey[i]*Z[j]*GammaI[j] -SigmaI**(X+Ksize*i+j);
   }
}

/*
┌──────────────────┐
│  次の状態を推定する  │
└──────────────────┘
*/
RungeKutta(X,mode,row,col)
   int mode,row,col;
   double X[row][col];
{
   double Xtmp[row][col],k0[row][col],k1[row][col],k2[row][col],k3[row][col];
   int i,j;
   NextState(X,k0,mode);
   for(j=0;j¡row;j++) for(i=0;i¡col;i++)
     Xtmp[j][i] = X[j][i] + 0.5*SamplingTime*k0[j][i];
   NextState(Xtmp,k1,mode);
   for(j=0;j¡row;j++) for(i=0;i¡col;i++)
     Xtmp[j][i] = X[j][i] + 0.5*SamplingTime*k1[j][i];
   NextState(Xtmp,k2,mode);
   for(j=0;j¡row;j++) for(i=0;i¡col;i++)
     Xtmp[j][i] = X[j][i] + SamplingTime*k2[j][i];
   NextState(Xtmp,k3,mode);
   for(j=0;j¡row;j++) for(i=0;i¡col;i++)
     X[j][i] += (k0[j][i]+2.0*k1[j][i]+2.0*k2[j][i]+k3[j][i])
                          * SamplingTime / 6.0;
}

/*
┌──────────────┐
│  パラメータ同定  │
└──────────────┘
*/
Adjust()
{
   int i,j;
   for(i=0   ; i ¡ M ;i++) Z[i]=Ey[i];
   for(i=M   ; i¡M+Nm;i++) Z[i]=Xm[i-M];
```

```
    for(i=M+Nm;i<Ksize;i++) Z[i]=Um[i-(M+Nm)];
    RungeKutta(Ki,ADJUSTMENT,R,Ksize);
    for(i=0;i<R;i++) for(j=0;j<Ksize;j++)
      Kp[i][j]= -Ey[i]*Z[j]*GammaP[j];
    for(i=0;i<R;i++) for(j=0;j<Ksize;j++) K[i][j] = Ki[i][j] + Kp[i][j];
    for(j=0;j<R;j++)-                    /* 各係数の導出 */
      for(i=0;i<M ;i++) Ke[j][i] = K[j][i];
      for(i=0;i<Nm;i++) Kx[j][i] = K[j][i+M];
      for(i=0;i<R ;i++) Ku[j][i] = K[j][i+M+Nm];
      ~
~
/*
 ┌──────────────┐
 │  SAC シミュレーション  │
 └──────────────┘
*/
SimulateSAC()
-
    int i,j;
    for(i=0;i<R;i++) Um[i]=StepRectangle(TIME+period/R*i*.5,amplitude);
    RungeKutta(Xm,CRITERION,Nm,1);      /* Xm を求める */
    Adjust();                           /* 調整則を求める */
    for(j=0;j<M;j++) Ey[j]=Yp[j]-Ym[j]; /* 誤差式 */
    for(i=0;i<R;i++)-
      for(j=0,Ue[i]=0;j<M ;j++) Ue[i] += Ke[i][j]*Ey[j];
      for(j=0,Ux[i]=0;j<Nm;j++) Ux[i] += Kx[i][j]*Xm[j];
      for(j=0,Uu[i]=0;j<R ;j++) Uu[i] += Ku[i][j]*Um[j];
      Up[i] = Uu[i] + Ux[i] + Ue[i]; /* プラント入力の決定 */
      ~
    RungeKutta(Xp,PLANT'MODEL,Np,1);  /* Xp を求める */
    TIME += SamplingTime;
~
/*
 ┌──────────┐
 │  主プログラム  │
 └──────────┘
*/
main(int argc,char *argv[])
-
    int   i,j;
    FILE *fp = (argc<1) ? fopen(argv[1],"w") : stdout;
    printf("Simple Adaptive Control - Simulation Program¥n");
    for(i=0;i<Np;i++)-
      for(j=0;j<Np;j++) printf("%8.4g ",Ap[i][j]);
      printf("¥n");
      ~
    while (TIME <= SamplingTime*MAX'COUNT)-
      SimulateSAC();
      fprintf(fp,"%8.4lf ",TIME);
      for(i=0;i<M;i++) fprintf(fp,"%8.4lf ",Yp[i]);
/*      for(i=0;i<R;i++) for(j=0;j<M;j++) fprintf(fp,"%8.4lf ",Ku[i][j]);*/
      fprintf(fp,"¥n");
      ~
    fclose(fp);
~
/*
```

*/

msacsim3.c

```
  /*


 */
/*
┌──────────────────────────────────────┐
│                                      │
│   Program : msacsim3.c (kappa)       │
│   Desc.   : MIMO-SAC Simulation Program  │
│   Creator : Budi Rachmanto 1996-11-14    │
│                                      │
│   多変数単純適応制御による制御シミュレーション   │
│                                      │
└──────────────────────────────────────┘
Plant :
  .      [ -1 1  0  0 ]        [1 0 0 0]
  Xp(t) = [ -2 -1  1  0 ] Xp(t) + [0 1 0 0] Up(t)
         [ 0 -2 -1  1 ]        [0 0 1 0]
         [ 0  0 -2 -1 ]        [0 0 0 1]
         [ 1  0  0  0 ]
  Yp     = [ 0  1  0  0 ] Xp(t)
```

```
        [ 0  0  1  0 ]
        [ 0  0  0  1 ]
 Criterion :
   Am  = -I*4
   Bm  = Cm = I*4
   Gm  = diag[1/(s+1), 1/(s+1), 1/(s+1), 1/(s+1)]
 Parameters :
   GammaI = diag[10^8I*2, 10^3I*4]
   GammaP = diag[10^6I*2, 10^2I*4]
*/
#include ¡stdio.h¿
#include ¡math.h¿
#include ¡time.h¿
#define PLANT*MODEL  0
#define CRITERION    1
#define ADJUST*Ke    2
#define ADJUST*Kx    3
#define ADJUST*Ku    4
#define YES          1
#define NO           0
/* #define PI          3.141592654 */
#define SamplingTime 0.01                /* サンプリング時間 (¡= .01) */
#define MAX*COUNT    104000                 /* 最大カウント数 */
#define period       40.                 /* 入力の周期 */
#define Np           4                   /* プラント次数 */
#define Nm           4                   /* 規範モデル次数 */
#define R            4                   /* 入力次数 */
#define M            4                   /* 出力次数 */
#define amplitude    1.0                   /* 矩形波の入力値 */
#define step         0.0                 /* ステップ入力 */
#define epsilon      .0000001
#define Sgm1         0.01                  /* 設計パラメータ */
#define Sgm2         0.05
#define kappa        1
/* Macros */
#define Ksize        (M+Nm+R)
#define notzero(x)   (fabs(x)¿epsilon) ? x : 0.
#define cont(i,j,k)  if(abs(i-j)¿=k) continue
static double Um[R],Up[R],                  /* 入力関数 */
      Uu[R],Ux[Nm],Ue[M],
      TIME = 0.0,                         /* 経過時間 */
      GammaI*Ke=10000,GammaI*Kx=10,GammaI*Ku=10,
      GammaP*Ke=10000,GammaP*Kx=10,GammaP*Ku=10,
      Ap[Np][Np]=- 3,-3, 0, 4,
                -1, 2, 3, 0,
                 0, 2, 4,-2,
                 5, 0,-1, 1",
      Xp[Np],Xm[Nm],
      SigmaI,EQuad,
      Ey[M]=-.0,.0,.0",Yp[M]=-0,0,0,0",Ym[M]=-0,0,0,0",
      Ke[R][M] ,KeI[R][M] ,             /* 調整パラメータ */
      Kx[R][Nm],KxI[R][Nm],               /* (Tuning Parameters) */
      Ku[R][R] ,KuI[R][R];
/*
 ┌──────────────────────────┐
 │   = Criterion*Path Generator =   │
 │                          │
 │  ┐┌┐┌┐┌┐┌┐┌┐┌┐┌┐┌┐┌┐  │
 │  └┘└┘└┘└┘└┘└┘└┘└┘└┘└┘  │
 │                          │
 │  規範モデル入力を決定する関数   │
 │  周期 period で矩形波を生成     │
 └──────────────────────────┘
*/
double StepRectangle(double t,double u)
{
   return( (fmod(t,period) ¡= period/2) ? u+step : -u+step );
}
/*
 ┌──────────────────────────┐
 │   状態方程式を用いて次の状態を出力   │
 └──────────────────────────┘
*/
NextState(X,Xdot,mode)
   int mode;
   double *X, *Xdot;
{
   int i,j;
   switch(mode)-
     case PLANT*MODEL :
       for(i=0;i¡Np;i++)-
         for(j=0,Xdot[i]=0;j¡Np;j++) Xdot[i]+=Ap[i][j]*X[j];
         Xdot[i]+=Up[i];
         Yp[i]=X[i];
       "
       break;
     case CRITERION :
       for(i=0;i¡Nm;i++) Xdot[i] = -(Ym[i]=X[i]) + Um[i];
       break;
     case ADJUST*Ke :
       for(i=0;i¡R;i++) for(j=0;j¡M;j++)-
         cont(i,j,kappa);
```

```
        *(Xdot+M*i+j) = -Ey[i]*Ey[j]*GammaI'Ke -SigmaI**(X+M*i+j);
        "
      break;
    case ADJUST'Kx :
      for(i=0;i<R;i++) for(j=0;j<Nm;j++)-
        cont(i,j,kappa);
        *(Xdot+Nm*i+j) = -Ey[i]*Xm[j]*GammaI'Kx -SigmaI**(X+Nm*i+j);
        "
      break;
    case ADJUST'Ku :
      for(i=0;i<R;i++) for(j=0;j<R;j++)-
        cont(i,j,kappa);
        *(Xdot+R*i+j) = -Ey[i]*Um[j]*GammaI'Ku -SigmaI**(X+R*i+j);
        "
  "
"

/*
┌──────────────────┐
|  次の状態を推定する  |
└──────────────────┘
*/
RungeKutta(X,mode,row,col)
  int mode,row,col;
  double X[row][col];
_
  double Xtmp[row][col],k0[row][col],k1[row][col],k2[row][col],k3[row][col];
  int i,j;
  NextState(X,k0,mode);
  for(j=0;j<row;j++) for(i=0;i<col;i++) -
    if(mode<=ADJUST'Ke) cont(i,j,kappa);
    Xtmp[j][i] = X[j][i] + 0.5*SamplingTime*k0[j][i];
    "
  NextState(Xtmp,k1,mode);
  for(j=0;j<row;j++) for(i=0;i<col;i++) -
    if(mode<=ADJUST'Ke) cont(i,j,kappa);
    Xtmp[j][i] = X[j][i] + 0.5*SamplingTime*k1[j][i];
    "
  NextState(Xtmp,k2,mode);
  for(j=0;j<row;j++) for(i=0;i<col;i++) -
    if(mode<=ADJUST'Ke) cont(i,j,kappa);
    Xtmp[j][i] = X[j][i] + SamplingTime*k2[j][i];
    "
  NextState(Xtmp,k3,mode);
  for(j=0;j<row;j++) for(i=0;i<col;i++) -
    if(mode<=ADJUST'Ke) cont(i,j,kappa);
    X[j][i] += (k0[j][i]+2.0*k1[j][i]+2.0*k2[j][i]+k3[j][i])
                        * SamplingTime / 6.0;
    "
"

/*
┌──────────────┐
|  パラメータ同定  |
└──────────────┘
*/
Adjust()
_
  int i,j;
  for(i=0,EQuad=0.;i<M;i++) EQuad += Ey[i]*Ey[i];
  SigmaI = Sgm1 * EQuad / (1.0 + EQuad) + Sgm2;
  RungeKutta(KeI,ADJUST'Ke,R,M);
  RungeKutta(KxI,ADJUST'Kx,R,Nm);
  RungeKutta(KuI,ADJUST'Ku,R,R);
  for(i=0;i<R;i++)-
    for(j=0;j<M;j++) -
      cont(i,j,kappa);
      Ke[i][j] = KeI[i][j] - Ey[i]*Ey[j]*GammaP'Ke;
      "
    for(j=0;j<Nm;j++)-
      cont(i,j,kappa);
      Kx[i][j] = KxI[i][j] - Ey[i]*Xm[j]*GammaP'Kx;
      "
    for(j=0;j<R;j++) -
      cont(i,j,kappa);
      Ku[i][j] = KuI[i][j] - Ey[i]*Um[j]*GammaP'Ku;
      "
    "
"

/*
┌──────────────────┐
|  SAC シミュレーション  |
└──────────────────┘
*/
SimulateSAC()
_
  int i,j;
  for(i=0;i<R;i++) Um[i]=StepRectangle(TIME+period/R*i*.5,amplitude);
  RungeKutta(Xm,CRITERION,Nm,1);      /* Xm を求める */
  Adjust();                           /* 調整則を求める */
  for(j=0;j<M;j++) Ey[j]=Yp[j]-Ym[j]; /* 誤差式 */
  for(i=0;i<R;i++)-
    for(j=0,Ue[i]=0;j<M ;j++) Ue[i] += Ke[i][j]*Ey[j];
    for(j=0,Ux[i]=0;j<Nm;j++) Ux[i] += Kx[i][j]*Xm[j];
    for(j=0,Uu[i]=0;j<R ;j++) Uu[i] += Ku[i][j]*Um[j];
```

```
    Up[i] = Uu[i] + Ux[i] + Ue[i]; /* プラント入力の決定 */
    ˜
  RungeKutta(Xp,PLANT'MODEL,Np,1);  /* Xp を求める */
  TIME += SamplingTime;
˜
/*
 ┌──────────┐
 │  主プログラム  │
 └──────────┘
*/
main(int argc,char *argv[])
_
  int  i,j,start,end;
  FILE *fp = (argc¿1) ? fopen(argv[1],"w") : stdout;
  printf("Simple Adaptive Control - Simulation Program¥n");
  for(i=0;i¡Np;i++)-
    for(j=0;j¡Np;j++) printf("%8.4g ",Ap[i][j]);
    printf("¥n");
    ˜
  while (TIME ¡ period)- /* 一周期目のデータをとる */
    SimulateSAC();
    if(TIME ¿= period) break;
    fprintf(fp,"%8.4lf ",TIME);
/*    for(i=0;i¡M;i++) fprintf(fp,"%8.4lf ",Yp[i]); */
    for(i=0;i¡M;i++) fprintf(fp,"%8.4lf ",Ey[i]);
/*    for(i=0;i¡R;i++) for(j=0;j¡M;j++) fprintf(fp,"%8.4lf ",Ke[i][j]);*/
    fprintf(fp,"¥n");
    ˜
  start=time(NULL);
  while (TIME ¡ SamplingTime*MAX'COUNT) SimulateSAC();
  end=time(NULL);
  printf("kappa=%d start=%d end=%d time=%d¥n",kappa,start,end,end-start);
  fclose(fp);
˜
/*
```

*/

Model.c

```
/*
 * Model.c (C-MEX)
 * Reference Model for MIMOSAC (MIMO Simple Adaptive Control)
 *
 * Syntax:  [sys, x0] = Model(t,x,u,flag,A,B)
 *
 * dot'x = -Ax + Bu
 *     y =  x
 *
 * Budi Rachmanto JAIST 19970128
 */
/*
 * The following #define is used to specify the name of this S-Function.
 */
#define S'FUNCTION'NAME Model
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
/*
 * Defines for easy access of the A, B matrices which are passed in
 */
#define ARG'A ssGetArg(S,0)
#define ARG'B ssGetArg(S,1)
/*
 * mdlInitializeSizes - initialize the sizes array
 *
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(S)
 SimStruct *S;
_
 ssSetNumContStates(    S,DYNAMICALLY'SIZED);/* number of continuous states */
 ssSetNumDiscStates(    S, 0);    /* number of discrete states */
 ssSetNumInputs(        S,DYNAMICALLY'SIZED);/* number of inputs */
 ssSetNumOutputs(       S,DYNAMICALLY'SIZED);/* number of outputs */
 ssSetDirectFeedThrough(S, 0);    /* direct feedthrough flag */
 ssSetNumSampleTimes(   S, 1);    /* number of sample times */
 ssSetNumInputArgs(     S, 2);    /* number of input arguments */
 ssSetNumRWork(         S, 0);    /* number of real work vector elements */
```

```
  ssSetNumIWork(          S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(          S, 0);    /* number of pointer work vector elements */
"

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 *
 * This function is used to specify the sample time(s) for your S-function.
 * If your S-function is continuous, you must specify a sample time of 0.0.
 * Sample times must be registered in ascending order.
 */
static void mdlInitializeSampleTimes(S)
    SimStruct *S;
_
    /*
     * This a purely continuous block, so I set the sample time
     * to 0.0.
     */
    ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
"

/*
 * mdlInitializeConditions - initialize the states
 *
 * In this function, you should initialize the continuous and discrete
 * states for your S-function block.  The initial states are placed
 * in the x0 variable.  You can also perform any other initialization
 * activities that your S-function may require.
 */
static void mdlInitializeConditions(x0, S)
    double *x0;
    SimStruct *S;
_
"

/*
 * mdlOutputs - compute the outputs
 *
 * In this function, you compute the outputs of your S-function
 * block.  The outputs are placed in the y variable.
 */
static void mdlOutputs(y, x, u, S, tid)
    double *y, *x, *u;
    SimStruct *S;
    int tid;
_
  int i;
  for(i=0; i<ssGetNumInputs(S); i++) y[i]=x[i];
"

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */
static void mdlUpdate(x, u, S, tid)
    double *x, *u;
    SimStruct *S;
    int tid;
_
"

/*
 * mdlDerivatives - compute the derivatives
 *
 * In this function, you compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */
static void mdlDerivatives(dx, x, u, S, tid)
    double *dx, *x, *u;
    SimStruct *S;
    int tid;
_
  double a, b;
  int i;
  a = mxGetPr(ARG'A)[0];
  b = mxGetPr(ARG'B)[0];
  for(i=0; i<ssGetNumInputs(S); i++) dx[i]= -a*x[i] + b*u[i];
"

/*
 * mdlTerminate - called when the simulation is terminated.
 *
 * In this function, you should perform any actions that are necessary
 * at the termination of a simulation.  For example, if memory was allocated
 * in mdlInitializeConditions, this is the place to free it.
 */
static void mdlTerminate(S)
    SimStruct *S;
_
"

#ifdef MATLAB'MEX'FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg'sfun.h"       /* Code generation registration function */
#endif
/*
```

*/

PFC.c

```
/*
 * PFC.c (C-MEX)
 * Parallel Feedforward Compensator for MIMOSAC
 *
 * Syntax:  [sys, x0] = PFC(t,x,u,flag,A,B)
 *
 * dot'x = -Ax + Bu
 *     y =   x
 *
 * Budi Rachmanto JAIST 19970128
 */
/*
 * The following #define is used to specify the name of this S-Function.
 */
#define S_FUNCTION_NAME PFC
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
/*
 * Defines for easy access of the A, B matrices which are passed in
 */
#define ARG_A ssGetArg(S,0)
#define ARG_B ssGetArg(S,1)
/*
 * mdlInitializeSizes - initialize the sizes array
 *
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(S)
  SimStruct *S;
{
  ssSetNumContStates(    S,DYNAMICALLY_SIZED);/* number of continuous states */
  ssSetNumDiscStates(    S, 0);    /* number of discrete states */
  ssSetNumInputs(        S,DYNAMICALLY_SIZED);/* number of inputs */
  ssSetNumOutputs(       S,DYNAMICALLY_SIZED);/* number of outputs */
  ssSetDirectFeedThrough(S, 0);    /* direct feedthrough flag */
  ssSetNumSampleTimes(   S, 1);    /* number of sample times */
  ssSetNumInputArgs(     S, 2);    /* number of input arguments */
  ssSetNumRWork(         S, 0);    /* number of real work vector elements */
  ssSetNumIWork(         S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(         S, 0);    /* number of pointer work vector elements */
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 *
 * This function is used to specify the sample time(s) for your S-function.
 * If your S-function is continuous, you must specify a sample time of 0.0.
 * Sample times must be registered in ascending order.
 */
static void mdlInitializeSampleTimes(S)
   SimStruct *S;
{
   /*
    * This a purely continuous block, so I set the sample time
    * to 0.0.
    */
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}
/*
 * mdlInitializeConditions - initialize the states
 *
 * In this function, you should initialize the continuous and discrete
 * states for your S-function block.  The initial states are placed
 * in the x0 variable.  You can also perform any other initialization
 * activities that your S-function may require.
 */
static void mdlInitializeConditions(x0, S)
   double *x0;
   SimStruct *S;
{
}
/*
 * mdlOutputs - compute the outputs
 *
 * In this function, you compute the outputs of your S-function
 * block.  The outputs are placed in the y variable.
 */
static void mdlOutputs(y, x, u, S, tid)
```

```
    double *y, *x, *u;
    SimStruct *S;
    int tid;
{
  int i;
  for(i=0; i<ssGetNumInputs(S); i++) y[i]=x[i];
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */
static void mdlUpdate(x, u, S, tid)
    double *x, *u;
    SimStruct *S;
    int tid;
{
}

/*
 * mdlDerivatives - compute the derivatives
 *
 * In this function, you compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */
static void mdlDerivatives(dx, x, u, S, tid)
    double *dx, *x, *u;
    SimStruct *S;
    int tid;
{
  double a, b;
  int i;
  a = mxGetPr(ARG'A)[0];
  b = mxGetPr(ARG'B)[0];
  for(i=0; i<ssGetNumInputs(S); i++) dx[i]= -a*x[i] + b*u[i];
}

/*
 * mdlTerminate - called when the simulation is terminated.
 *
 * In this function, you should perform any actions that are necessary
 * at the termination of a simulation.  For example, if memory was allocated
 * in mdlInitializeConditions, this is the place to free it.
 */
static void mdlTerminate(S)
    SimStruct *S;
{
}

#ifdef MATLAB'MEX'FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg'sfun.h"       /* Code generation registration function */
#endif
/*
```

*/

MIMOSAC.c

```
/*
 *
 * MIMOSAC.c (C-MEX)
 * Multi Input Multi Output SAC
 *
 * Syntax:  [sys, x0] = MIMOSAC(t,x,u,flag,SigmaK0,GammaKI,GammaKP,NOut)
 *
 * Budi Rachmanto JAIST 19970128
 */
#define S'FUNCTION'NAME MIMOSAC
#define SigmaK0 ssGetArg(S,0)
#define GammaKI ssGetArg(S,1)
#define GammaKP ssGetArg(S,2)
#define NOut ssGetArg(S,3)
#include <math.h>
#include "simstruc.h"
static void mdlInitializeSizes(S)
  SimStruct *S;
{
  int m=mxGetPr(NOut)[0];
  ssSetNumContStates(    S, DYNAMICALLY'SIZED);
                          /* number of continuous states (KI) */
  ssSetNumDiscStates(    S, 0);    /* number of discrete states */
  ssSetNumInputs(        S, DYNAMICALLY'SIZED);
```

```
                                /* number of inputs (Ey,Xm,Um) */
  ssSetNumOutputs(       S, m);    /* number of outputs */
  ssSetDirectFeedThrough(S, 1);    /* direct feedthrough flag (KP) */
  ssSetNumSampleTimes(   S, 1);    /* number of sample times */
  ssSetNumInputArgs(     S, 4);    /* number of input arguments */
  ssSetNumRWork(         S, 0);    /* number of real work vector elements */
  ssSetNumIWork(         S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(         S, 0);    /* number of pointer work vector elements */
"
static void mdlInitializeSampleTimes(S)
  SimStruct *S;
_
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
"
static void mdlInitializeConditions(x0,S)
  double *x0;
  SimStruct *S;
_
"
static void mdlOutputs(U'l, KI, z, S, tid)
   double *U'l, *KI, *z; /* z=[Ey,Xm,Um]^T */
   SimStruct *S;
   int tid;
_
  int i,j;
  int m=ssGetNumOutputs(S);
  int n=ssGetNumInputs(S);
  double *GammaP;
  GammaP  = mxGetPr(GammaKP);
  for(i=0;i<m;i++)-
    U'l[i] = 0.;
    for(j=0;j<m;j++)  U'l[i] += (*(KI+i*n+j) - z[i]*z[j]*GammaP[0])*z[j];
    for(j=m;j<n-m;j++)U'l[i] += (*(KI+i*n+j) - z[i]*z[j]*GammaP[1])*z[j];
    for(j=n-m;j<n;j++)U'l[i] += (*(KI+i*n+j) - z[i]*z[j]*GammaP[2])*z[j];
   "
"
/*
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */
static void mdlUpdate(x, u, S, tid)
   double *x, *u;
   SimStruct *S;
   int tid;
_
"
static void mdlDerivatives(dotKI, KI, z, S, tid)
   double *dotKI, *KI, *z; /* z=[Ey,Xm,Um]^T */
   SimStruct *S;
   int tid;
_
  int i,j;
  int m=ssGetNumOutputs(S);
  int n=ssGetNumInputs(S);
  double SigmaK, EyQuad=0., *GammaI;
  GammaI = mxGetPr(GammaKI);
  for(i=0; i<m; i++) EyQuad += z[i]*z[i];
/*
  SigmaK = mxGetPr(SigmaK0)[0]*EyQuad/(1+EyQuad) + mxGetPr(SigmaK0)[0];
*/
  SigmaK = mxGetPr(SigmaK0)[0]*EyQuad/(1+EyQuad);
/*
  for(i=0; i<m; i++) for(j=0; j<n; j++)
    *(dotKI+i*n+j) = -z[i]*z[j]*GammaI[j] - SigmaK**(KI+i*n+j);
*/
  for(i=0; i<m; i++)-
    for(j=0; j<m; j++)
      *(dotKI+i*n+j) = -z[i]*z[j]*GammaI[0] - SigmaK**(KI+i*n+j);
    for(j=m; j<n-m; j++)
      *(dotKI+i*n+j) = -z[i]*z[j]*GammaI[1] - SigmaK**(KI+i*n+j);
    for(j=n-m; j<n; j++)
      *(dotKI+i*n+j) = -z[i]*z[j]*GammaI[2] - SigmaK**(KI+i*n+j);
    "
"
static void mdlTerminate(S)
   SimStruct *S;
_
"
#ifdef MATLAB'MEX'FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg'sfun.h"       /* Code generation registration function */
#endif
/*
```

*/

NLC.c

```
/*
 * NLC.c (C-MEX)
 * NonLinear Compensator for MIMOSAC
 *
 * Syntax:  [sys, x0] = NLC(t,x,u,flag,SigmaH,Epsilon,GammaHI,GammaHP,NOut)
 *
 * Budi Rachmanto JAIST 19970128
 */
#define S_FUNCTION_NAME NLC
#define Sigma0 ssGetArg(S,0)
#define Epsilon ssGetArg(S,1)
#define GammaHI ssGetArg(S,2)
#define GammaHP ssGetArg(S,3)
#define NOut ssGetArg(S,4)
#define sgn(x) (double)((x>0.)-(x<0.))
#include <math.h>
#include "simstruc.h"
static void mdlInitializeSizes(S)
  SimStruct *S;
{
  int m=mxGetPr(NOut)[0];
  ssSetNumContStates(     S, 2*m);  /* number of continuous states (HI0,HI1) */
  ssSetNumDiscStates(     S, 0);    /* number of discrete states */
  ssSetNumInputs(         S, 3*m);  /* number of inputs (Ey,Ya,dot_Ya) */
  ssSetNumOutputs(        S, m);    /* number of outputs */
  ssSetDirectFeedThrough(S, 1);    /* direct feedthrough flag (HP) */
  ssSetNumSampleTimes(    S, 1);    /* number of sample times */
  ssSetNumInputArgs(      S, 5);    /* number of input arguments */
  ssSetNumRWork(          S, 0);    /* number of real work vector elements */
  ssSetNumIWork(          S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(          S, 0);    /* number of pointer work vector elements */
}

static void mdlInitializeSampleTimes(S)
  SimStruct *S;
{
    ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
}

static void mdlInitializeConditions(x0, S)
  double *x0;
  SimStruct *S;
{
}

static void mdlOutputs(U_R, HI, u, S, tid)
    double *U_R, *HI, *u; /* u=(Ey,Ya,dot_Ya) */
    SimStruct *S;
    int tid;
{
  int i,j, m=ssGetNumOutputs(S);
  double H0,H1,Hz,Hze,epsilon,NormYh,*GammaP;
  epsilon = mxGetPr(Epsilon)[0];
  GammaP  = mxGetPr(GammaHP);
  for(i=0; i<m; i++){
    NormYh = sqrt(u[m+i]*u[m+i] + u[m+m+i]*u[m+m+i]);
    H0     = HI[i]   + sqrt(u[i]*u[i])      *GammaP[0];
    H1     = HI[i+m] + sqrt(u[i]*u[i])*NormYh*GammaP[1];
    Hz     = H0 + H1*NormYh;
    Hze    = fabs(Hz*u[i]);
    if(Hze<=epsilon){
      U_R[i] = -Hz*Hz*u[i]/epsilon;
    } else {
      U_R[i] = -Hz*sgn(u[i]);
    }
  }
}

/*
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */
static void mdlUpdate(x, u, S, tid)
    double *x, *u;
    SimStruct *S;
    int tid;
{
}

static void mdlDerivatives(dotHI, HI, u, S, tid)
    double *dotHI, *HI, *u; /* u=(Ey,Ya,dot_Ya) */
    SimStruct *S;
    int tid;
{
  int i, m=ssGetNumOutputs(S);
  double SigmaH, NormYh, *GammaI;
  for(i=0; i<m; i++){
    SigmaH = mxGetPr(Sigma0)[0]*u[i]*u[i]/(1+u[i]*u[i]);
    NormYh = sqrt(u[m+i]*u[m+i] + u[m+m+i]*u[m+m+i]);
```

```
    GammaI = mxGetPr(GammaHI);
    dotHI[i]   = fabs(u[i])       *GammaI[0] - SigmaH*HI[i];
    dotHI[i+m] = fabs(u[i])*NormYh*GammaI[1] - SigmaH*HI[i+m];
  "
"
static void mdlTerminate(S)
    SimStruct *S;
_
"
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
/*
```

*/

PSAC.c

/*

```
 * PSAC.c (C-MEX)
 * Multi Input Multi Output SAC (Parallelized SISO-SAC)
 *
 * Syntax:  [sys, x0] = PSAC(t,x,u,flag,SigmaK0,GammaKI,GammaKP,NOut)
 *
 * Budi Rachmanto JAIST 19970128
 */
#define S_FUNCTION_NAME PSAC
#define SigmaK0 ssGetArg(S,0)
#define GammaKI ssGetArg(S,1)
#define GammaKP ssGetArg(S,2)
#define NOut ssGetArg(S,3)
#include ¡math.h¿
#include "simstruc.h"
static void mdlInitializeSizes(S)
  SimStruct *S;
_
  int m=mxGetPr(NOut)[0];
  ssSetNumContStates(    S, DYNAMICALLY_SIZED);
                          /* number of continuous states (KI) */
  ssSetNumDiscStates(    S, 0);    /* number of discrete states */
  ssSetNumInputs(        S, DYNAMICALLY_SIZED);
                          /* number of inputs (Ey,Xm,Um) */
  ssSetNumOutputs(       S, m);    /* number of outputs */
  ssSetDirectFeedThrough(S, 1);    /* direct feedthrough flag (KP) */
  ssSetNumSampleTimes(   S, 1);    /* number of sample times */
  ssSetNumInputArgs(     S, 4);    /* number of input arguments */
  ssSetNumRWork(         S, 0);    /* number of real work vector elements */
  ssSetNumIWork(         S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(         S, 0);    /* number of pointer work vector elements */
"
static void mdlInitializeSampleTimes(S)
  SimStruct *S;
_
    ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
"
static void mdlInitializeConditions(x0,S)
  double *x0;
  SimStruct *S;
_
"
static void mdlOutputs(U_l, KI, z, S, tid)
    double *U_l, *KI, *z; /* z=[Ey,Xm,Um]^T */
    SimStruct *S;
    int tid;
_
  int i;
  int m=ssGetNumOutputs(S);
  double *GammaP;
  GammaP  = mxGetPr(GammaKP);
  for(i=0;i;m;i++)
    U_l[i] = (*(KI+i)     -z[i]*z[i]     *GammaP[0])*z[i]
           + (*(KI+i+m)   -z[i]*z[i+m]   *GammaP[1])*z[i+m]
           + (*(KI+i+m+m)-z[i]*z[i+m+m]*GammaP[2])*z[i+m+m];
"
/*
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
```

```
 */
static void mdlUpdate(x, u, S, tid)
   double *x, *u;
   SimStruct *S;
   int tid;
{
}

static void mdlDerivatives(dotKI, KI, z, S, tid)
   double *dotKI, *KI, *z; /* z=[Ey,Xm,Um]^T */
   SimStruct *S;
   int tid;
{
  int i;
  int m=ssGetNumOutputs(S);
  int n=ssGetNumInputs(S);
  double SigmaK, EyQuad=0., *GammaI;
  GammaI = mxGetPr(GammaKI);
/*
  for(i=0; i;m; i++) EyQuad += z[i]*z[i];
  SigmaK = mxGetPr(SigmaK0)[0]*EyQuad/(1+EyQuad);
*/
  SigmaK = mxGetPr(SigmaK0)[0];
  for(i=0;i;m;i++)-
    *(dotKI+i)     = -z[i]*z[i]     *GammaI[0] - SigmaK**(KI+i);
    *(dotKI+i+m)   = -z[i]*z[i+m]   *GammaI[1] - SigmaK**(KI+i+m);
    *(dotKI+i+m+m)= -z[i]*z[i+m+m]*GammaI[2] - SigmaK**(KI+i+m+m);
  }
}

static void mdlTerminate(S)
   SimStruct *S;
{
}

#ifdef MATLAB'MEX'FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg'sfun.h"       /* Code generation registration function */
#endif
/*
```

*/

RNLC.c

```
/*


 * RNLC.c (C-MEX)
 * Reduced NonLinear Compensator for MIMOSAC
 *
 * Syntax:  [sys, x0] = RNLC(t,x,u,flag,SigmaH,Epsilon,GammaHI,GammaHP,NOut)
 *
 * Budi Rachmanto JAIST 19970128
 */
#define S'FUNCTION'NAME RNLC
#define Sigma0 ssGetArg(S,0)
#define Epsilon ssGetArg(S,1)
#define GammaHI ssGetArg(S,2)
#define GammaHP ssGetArg(S,3)
#define NOut ssGetArg(S,4)
#define sgn(x) (double)((x¿0.)-(x;0.))
#include ;math.h¿
#include "simstruc.h"
static void mdlInitializeSizes(S)
  SimStruct *S;
{
  int m=mxGetPr(NOut)[0];
  ssSetNumContStates(    S, m);    /* number of continuous states (HI) */
  ssSetNumDiscStates(    S, 0);    /* number of discrete states */
  ssSetNumInputs(        S, m);    /* number of inputs (Ey) */
  ssSetNumOutputs(       S, m);    /* number of outputs */
  ssSetDirectFeedThrough(S, 1);    /* direct feedthrough flag (HP) */
  ssSetNumSampleTimes(   S, 1);    /* number of sample times */
  ssSetNumInputArgs(     S, 5);    /* number of input arguments */
  ssSetNumRWork(         S, 0);    /* number of real work vector elements */
  ssSetNumIWork(         S, 0);    /* number of integer work vector elements */
  ssSetNumPWork(         S, 0);    /* number of pointer work vector elements */
}

static void mdlInitializeSampleTimes(S)
  SimStruct *S;
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}

static void mdlInitializeConditions(x0,S)
```

```
  double *x0;
  SimStruct *S;
_"
static void mdlOutputs(U'R, HI, u, S, tid)
   double *U'R, *HI, *u; /* u=(Ey,Ya,dot'Ya) */
   SimStruct *S;
   int tid;
_
  int i,j, m=ssGetNumOutputs(S);
  double H0,H1,Hz,Hze,epsilon,NormYh,*GammaP;
  epsilon = mxGetPr(Epsilon)[0];
  GammaP  = mxGetPr(GammaHP);
  for(i=0; i;m; i++)-
    Hz     = HI[i]   + sqrt(u[i]*u[i])       *GammaP[0];
    Hze    = fabs(Hz*u[i]);
    if(Hze;=epsilon) U'R[i] = -Hz*Hz*u[i]/epsilon;
    else             U'R[i] = -Hz*sgn(u[i]);
  "
"
/*
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */
static void mdlUpdate(x, u, S, tid)
   double *x, *u;
   SimStruct *S;
   int tid;
_"
static void mdlDerivatives(dotHI, HI, u, S, tid)
   double *dotHI, *HI, *u; /* u=(Ey,Ya,dot'Ya) */
   SimStruct *S;
   int tid;
_
  int i, m=ssGetNumOutputs(S);
  double SigmaH, NormYh, *GammaI;
  for(i=0; i;m; i++)-
    SigmaH   = mxGetPr(Sigma0)[0];
    GammaI   = mxGetPr(GammaHI);
    dotHI[i] = fabs(u[i])*GammaI[0] - SigmaH*HI[i];
  "
"
static void mdlTerminate(S)
   SimStruct *S;
_"
#ifdef MATLAB'MEX'FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg'sfun.h"       /* Code generation registration function */
#endif
/*
```

*/