

倒立振子の安定化制御

渡部僚太

2018年7月2日

目 次

第 1 章 はじめに	1
1.1 実験目的	1
1.2 制御対象	2
1.2.1 倒立振子系	2
1.2.2 観測出力と操作入力	3
第 2 章 モデリング	5
2.1 倒立振子のモデリング	5
2.1.1 状態方程式	5
2.1.2 観測方程式	10
2.2 倒立振子のパラメータの同定	11
2.2.1 m と l の測定	11
2.2.2 a の測定	11
2.2.3 J と c の測定	12
2.2.4 M と f の測定	14
2.2.5 c_1 と c_2 の測定	18
2.2.6 同定結果	18
2.3 パラメータの検証	18
2.3.1 振子の自由振動シミュレーション	19
2.3.2 台車のステップ応答による方法のシミュレーション	20
2.3.3 台車のフィードバック応答シミュレーション	21
2.4 設計（線形）モデルの決定	21
第 3 章 制御系設計	23
3.1 システム解析	23
3.1.1 安定性	23
3.1.2 可制御性	23
3.1.3 可観測性	24
3.2 状態フィードバックの設計	24
3.3 最小次元オブザーバの設計	25
3.4 コントローラの離散化	26
3.5 振り上げ制御及び安定化の実現	27

第 4 章 シミュレーション	29
4.1 重み行列の変更による制御性能評価	29
4.2 オブザーバの極の変更による制御性能評価	31
4.3 サンプリング周期の変更による制御性能評価	34
4.4 振り上げ制御及び安定化に対する制御性能評価	36
第 5 章 実験	39
5.1 安定化制御実験	39
5.2 目標値の変更実験	41
5.2.1 重み行列の違いによる実験結果の比較	41
5.2.2 オブザーバの極の違いによる実験結果の比較	42
5.2.3 サンプリング周期の違いによる実験結果の比較	43
5.2.4 シミュレーション結果と実験結果の比較	44
5.3 振り上げ制御及び安定化実験	49
第 6 章 おわりに	51
第 7 章 感想	53
付 錄 A プログラム	57
A.1 非線形モデル	57
A.2 線形モデル	57
A.3 システム解析	58
A.4 状態フィードバック	58
A.5 最小次元オブザーバ	59
A.6 コントローラの離散化	59
A.7 シミュレーション	59
A.8 安定化制御及び目標値変更実験	63
A.9 振り上げ制御及び安定化実験	67

図 目 次

1.1 倒立振子系	2
1.2 倒立振子系(写真)	3
2.1 数式モデル導出のための参考図	5
2.2 パラメータ a の決定	11
2.3 J と c の測定	13
2.4 台車のステップ応答	14
2.5 台車のフィードバック応答	15
2.6 フィードバック制御系のブロック線図	16
2.7 飽和器の有無によるフィードバック入力の比較	17
2.8 振子の自由振動シミュレーション	19
2.9 台車のステップ応答シミュレーションと実験結果の比較	20
2.10 フィードバックによる方法のシミュレーションと実験結果の比較	21
3.1 状態フィードバックを含めたブロック線図	25
3.2 最小次元オブザーバーを含めたブロック線図	26
3.3 離散器(0次ホールド)を含めたブロック線図	26
3.4 振り上げ制御及び安定化制御を行うためのブロック線図	28
4.1 重み行列での比較結果(台車位置)	30
4.2 重み行列での比較結果(振子角度)	30
4.3 オブザーバーの極での比較結果(台車位置)	32
4.4 オブザーバーの極での比較結果(振子角度)	32
4.5 オブザーバーの極での推定誤差の比較(台車の速度)	33
4.6 オブザーバーの極での推定誤差の比較(振子の角速度)	33
4.7 サンプリング周期での比較結果(台車位置)	34
4.8 サンプリング周期での比較結果(振子角度)	35
4.9 サンプリング周期での推定誤差の比較(台車の速度)	35
4.10 サンプリング周期での推定誤差の比較(振子の各速度)	36
4.11 振り上げ制御のシミュレーション比較結果	37
5.1 安定化制御実験結果(台車位置)	39

5.2 安定化制御実験結果（振子角度）	40
5.3 重み行列の違いによる実験結果の比較	41
5.4 オブザーバの極の違いによる実験結果の比較	42
5.5 サンプリング周期の違いによる実験結果の比較	43
5.6 比較結果その 1 (左図が r , 右図が θ)	44
5.7 比較結果その 2(左図が r , 右図が θ)	45
5.8 比較結果その 3(左図が r , 右図が θ)	45
5.9 比較結果その 4(左図が r , 右図が θ)	45
5.10 比較結果その 5(左図が r , 右図が θ)	46
5.11 比較結果その 6(左図が r , 右図が θ)	46
5.12 比較結果その 7(左図が r , 右図が θ)	46
5.13 比較結果その 8(左図が r , 右図が θ)	47
5.14 比較結果その 9(左図が r , 右図が θ)	47
5.15 比較結果その 10(左図が r , 右図が θ)	47
5.16 比較結果その 11(左図が r , 右図が θ)	48
5.17 比較結果その 12(左図が r , 右図が θ)	48
5.18 k の違いによる実験結果の比較	49
5.19 比較結果 (Pattern1)	49
5.20 比較結果 (Pattern2)	50
5.21 比較結果 (Pattern3)	50

表 目 次

2.1 同定したパラメータの一覧	18
4.1 重み行列の変更パターン	29
4.2 オブザーバの極の変更パターン	31
4.3 サンプリング周期の変更パターン	34
4.4 振り上げ後の安定化制御の際に用いるパラメータの組	36
4.5 振り上げ制御に用いるパラメータの組	37
5.1 安定化制御実験で使用したパラメータの組	39
5.2 パターンとパラメータの対応表	41
5.3 パターンとパラメータの対応表	42
5.4 パターンとパラメータの対応表	43
5.5 図に対応するパラメータの組	44
5.6 実験に用いたパラメータの組	49

第1章 はじめに

1.1 実験目的

本実験の目的は、倒立振子の安定化制御の制御系の設計を状態空間法を用いて行うことにより、線形時不变システムを設計することである。[1] 具体的には以下の

- 振子を垂直上向きに配置した状態から実験を開始し、安定化制御を行う。また、このとき十数度角度を傾けても安定化制御を行えるようにする。(不安定平衡点の安定化)
- 安定化制御を行っている状態でパルス入力を加え、台車の位置が変わっても安定化制御を行えるようにする。
- 振子を下向きに配置した状態から振り上げ、安定化制御を行えるようにする。

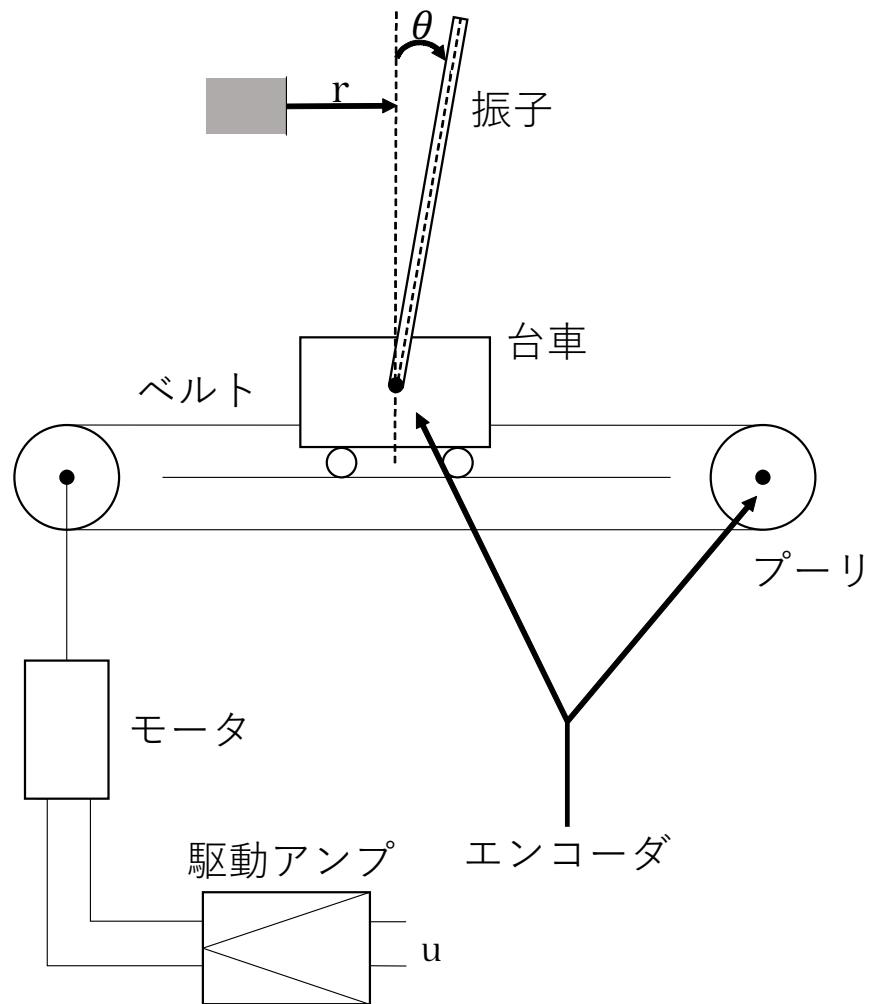
を達成することが目的である。

不安定平衡点とは、倒立振子系における平衡点の1つである。今回実験で使用する倒立振子系には平衡点が2つ存在する。1つは棒が鉛直線に沿って垂れ下がった状態、もう1つは棒が鉛直線に沿って倒立した状態である。前者は振子を揺らした場合、時間が立てば止まる安定平衡点である。後者は振子を揺らした場合、そのまま振子が真っ逆さまに落ちていく不安定平衡点である。

1.2 制御対象

1.2.1 倒立振子系

本実験で用いる倒立振子系の図を以下に載せる。また、実際に実験で用いる倒立振子系の写真も載せる。



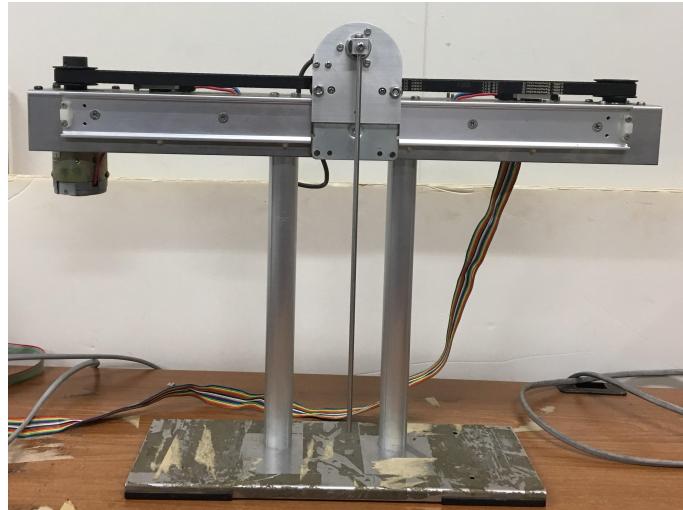


図 1.2: 倒立振子系 (写真)

図 1.1 は倒立振子系を表す図である。モノレールの上に台車が置かれ、台車上のモノレールと直角な軸に 1 本の棒が取り付けられ、棒はその軸まわりに自由に回転できる。台車はベルトとブーリーを介して、モータにより駆動され、モノレール上を走行できる。すなわち、棒（振子）は鉛直線とモノレールにより定まる平面に拘束されて、台車によって動かされるようになっている。[2]

図 1.2 は実機の倒立振子である。駆動アンプは写っていないが、それ以外の部品は図 1.1 と同じように確認できる。この倒立振子系は机に固定されているわけではなく、机の上に置かれているだけである。

1.2.2 観測出力と操作入力

倒立振子系の観測出力として、エンコーダにより、つぎの 2 つが測定できる。

- 台車の基準位置から変位 r に比例する電圧 y_1
- 棒の鉛直線となす角度 θ に比例する電圧 y_2

一方、操作入力は、つぎのものである。

- モータの駆動アンプの入力電圧 u

ここで、モータにより駆動される台車には、 u に比例した駆動力が働くものとする。[2] また、入力電圧の限界値は $-15 < u < 15$ である。

第2章 モデリング

2.1 倒立振子のモデリング

実験目的を達成する制御システムを設計するためには、倒立振子系について、状態方程式と観測方程式から成る数式モデルを導出する。

2.1.1 状態方程式

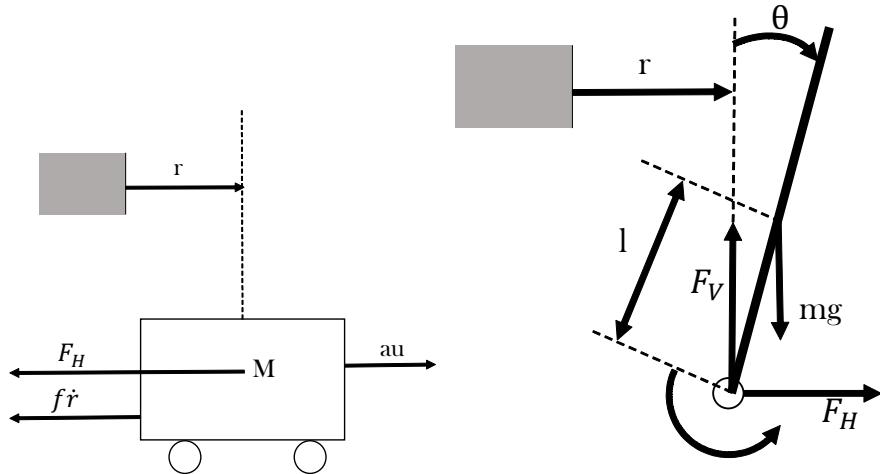


図 2.1: 数式モデル導出のための参考図

図 2.1 を参考に各運動方程式を導出すると、
台車の運動方程式は

$$M\ddot{r} = au - F_H - f\dot{r} \quad (2.1)$$

倒立振子の回転の運動方程式は

$$J\ddot{\theta} = lF_V \sin \theta - lF_H \cos \theta - c\dot{\theta} \quad (2.2)$$

倒立振子の水平方向の運動方程式は

$$m \frac{d^2}{dt^2} (r + l \sin \theta) = F_H \quad (2.3)$$

倒立振子の垂直方向の運動方程式は

$$m \frac{d^2}{dt^2} (l \cos \theta) = F_V - mg \quad (2.4)$$

となる。

ここで各式の導出過程を述べる。図2.1より台車の運動方程式は、振り子からの水平抗力 F_H を考慮してニュートンの第二法則より (2.1) 式を導くことができる。ただし、 M は台車の質量、 f は台車の摩擦係数、 a は駆動アンプへの入力電圧から台車への駆動までのゲイン、 u はモータの駆動アンプへの入力電圧、 r は台車の基準位置からの変位である。同様にニュートンの第二法則を用いることでそれぞれの方向における (2.3), (2.4) 式の運動方程式を導くことができる。ただし、 m は振り子の質量、 l は回転軸・重心間の距離、 g は重力加速度、 F_V は振り子が台車から受ける垂直抗力である。また、 θ は鉛直上向きを $\theta = 0$ としたときの角度である。

最後に (2.2) 式は回転に対する運動方程式を考えることで上記と同様に求めることができる。ただし、 J は重心回りの慣性モーメント、 c は回転軸摩擦係数である。

いま、4つの状態変数から成るベクトル、すなわち状態 x を

$$x = \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix}$$

のように定義し、(2.1) 式,(2.2) 式,(2.3) 式,(2.4) 式から倒立振子系の非線形状態方程式を求める。

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{r} \\ \dot{\theta} \\ \ddot{r} \\ \ddot{\theta} \end{bmatrix} \quad (2.5)$$

ここで (2.3) 式より F_H を、(2.4) 式より F_V を求めると

$$\begin{aligned} F_H &= m \frac{d^2}{dt^2} r + ml \frac{d^2}{dt^2} \sin \theta \\ &= m\ddot{r} + ml(-\dot{\theta}^2 \sin \theta + \ddot{\theta} \cos \theta) \end{aligned} \quad (2.6)$$

$$\begin{aligned} F_V &= mg + m \frac{d^2}{dt^2}(l \cos \theta) \\ &= mg + ml(-\dot{\theta}^2 \cos \theta - \ddot{\theta} \sin \theta) \end{aligned} \quad (2.7)$$

である。

(2.6) 式を (2.1) 式に代入すると

$$(M+m)\ddot{r} + ml\ddot{\theta} \cos \theta - ml\dot{\theta}^2 \sin \theta + f\dot{r} = au \quad (2.8)$$

である。

(2.6) 式と (2.7) 式を (2.2) 式に代入すると

$$(J+ml^2)\ddot{\theta} + ml\ddot{r} \cos \theta - mgl \sin \theta + c\dot{\theta} = 0 \quad (2.9)$$

である。

(2.8) 式、(2.12) 式を行列表現すると

$$\begin{bmatrix} (M+m)\ddot{r} + (ml \cos \theta)\ddot{\theta} + (-ml \sin \theta) + f\dot{r} = au \\ (ml \cos \theta)\ddot{r} + (J+ml^2)\ddot{\theta} - mgl \sin \theta + c\dot{\theta} = 0 \end{bmatrix}$$

$$\begin{bmatrix} M+m & ml \cos \theta \\ ml \cos \theta & J+ml^2 \end{bmatrix} \begin{bmatrix} \ddot{r} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -ml\dot{\theta}^2 \sin \theta + f\dot{r} \\ mgl \sin \theta + c\dot{\theta} \end{bmatrix} = \begin{bmatrix} au \\ 0 \end{bmatrix}$$

$\begin{pmatrix} M+m & ml \cos \theta \\ ml \cos \theta & J+ml^2 \end{pmatrix}$ を K と置いて右辺に逆行列としてかけると

$$\begin{bmatrix} \ddot{r} \\ \ddot{\theta} \end{bmatrix} = K^{-1} \begin{bmatrix} au - f\dot{r} + ml\dot{\theta} \sin \theta \\ mgl \sin \theta - c\dot{\theta} \end{bmatrix}$$

よって以上から (2.5) 式は

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{r} \\ \dot{\theta} \\ K^{-1} \begin{bmatrix} -f\dot{r} + ml\dot{\theta} \sin \theta + au \\ mgl \sin \theta - c\dot{\theta} \end{bmatrix} \end{bmatrix} \quad (2.10)$$

となる。ただし、 K は

$$K = \begin{bmatrix} M+m & ml \cos \theta \\ ml \cos \theta & J+ml^2 \end{bmatrix} \quad (2.11)$$

である。よって倒立振子系の非線形状態方程式は(2.10)式のように得られる。

ところで、倒立振子系については、その制御目的から、不安定平衡点 $x = 0$ の近傍での挙動を表す状態方程式を知れば十分である。そこで、この基準状態まわりで一時近似された状態方程式を求めるを考える。

(2.10)式に一次近似のテイラー展開を施すと、

$$f(x, u) = f(0, 0) + \frac{\partial f}{\partial x} \Big|_{x=0, u=0} (x - 0) + \frac{\partial f}{\partial u} \Big|_{x=0, u=0} (u - 0) \quad (2.12)$$

となる。つまり、(2.12)を計算すれば求めたい状態方程式を得ることができる。

(2.12)式において、

$$A = \frac{\partial f}{\partial x} \Big|_{x=0, u=0}, B = \frac{\partial f}{\partial u} \Big|_{x=0, u=0}$$

とすると、(2.12)式は以下のように計算できる。

$$f(x, u) = Ax + Bu \quad (2.13)$$

ここで、一時近似を施したので、 θ を微小範囲と考えることができ、
 $\sin \theta \simeq \theta, \cos \theta \simeq 1, \dot{\theta}^2 \simeq 0$ のように近似できる。

以上の近似から(2.10),(2.11)式は

$$f(x, u) = \begin{bmatrix} \dot{r} \\ \dot{\theta} \\ K'^{-1} \begin{bmatrix} au - f\dot{r} \\ mlg\theta - c\dot{\theta} \end{bmatrix} \end{bmatrix} \quad (2.14)$$

$$K' = \begin{bmatrix} M+m & ml \\ ml & J+ml^2 \end{bmatrix} \quad (2.15)$$

となる。ここで、(2.14)式の3行目を a_1 と置き、4行目を a_2 と置く。

(2.14)、(2.15) 式を用いて (2.13) 式の A、B を計算する。

$$A = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial \dot{r}}{\partial r} & \frac{\partial \dot{r}}{\partial \theta} & \frac{\partial \dot{r}}{\partial \dot{r}} & \frac{\partial \dot{r}}{\partial \dot{\theta}} \\ \frac{\partial \dot{\theta}}{\partial r} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \dot{r}} & \frac{\partial \dot{\theta}}{\partial \dot{\theta}} \\ \frac{\partial a_1}{\partial r} & \frac{\partial a_1}{\partial \theta} & \frac{\partial a_1}{\partial \dot{r}} & \frac{\partial a_1}{\partial \dot{\theta}} \\ \frac{\partial a_2}{\partial r} & \frac{\partial a_2}{\partial \theta} & \frac{\partial a_2}{\partial \dot{r}} & \frac{\partial a_2}{\partial \dot{\theta}} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & K'^{-1}(-f) & 0 \\ 0 & K'^{-1}(mgl) & 0 & K'^{-1}(-c) \end{bmatrix}$$

$$B = \frac{\partial f}{\partial u} = \begin{bmatrix} \frac{\partial \dot{r}}{\partial u} \\ \frac{\partial \dot{\theta}}{\partial u} \\ \frac{\partial a_1}{\partial u} \\ \frac{\partial a_2}{\partial u} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ K'^{-1}a \\ 0 \end{bmatrix}$$

以上から線形状態方程式は

$$\dot{x} = Ax + Bu$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & K'^{-1}(-f) & 0 \\ 0 & K'^{-1}(mgl) & 0 & K'^{-1}(-c) \end{bmatrix} \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ K'^{-1}au \\ 0 \end{bmatrix} \quad (2.16)$$

となる。ただし、 K' は (2.15) 式である。上式の線形状態方程式は鉛直上向きを $\theta = 0$ としたときの状態方程式である。鉛直下向きを $\theta = 0$ とした場合は (2.11) 式の三角関数内の θ に $+\pi$ すればよいので

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{r} \\ \dot{\theta} \\ K^{-1} \begin{bmatrix} -f\dot{r} - ml\ddot{\theta} \sin \theta + au \\ -mgl \sin \theta - c\dot{\theta} \end{bmatrix} \end{bmatrix} \quad (2.17)$$

となる。ただし、 K は

$$K = \begin{bmatrix} M+m & -ml \cos \theta \\ -ml \cos \theta & J+ml^2 \end{bmatrix} \quad (2.18)$$

である。

振子の角度を鉛直上向きを $\theta = 0$ としたときの状態方程式を線形化したときと同様に (2.17),(2.18) 式を線形化すると線形状態方程式は

$$\begin{aligned} \dot{x} &= Ax + Bu \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & K'^{-1}(-f) & 0 \\ 0 & K'^{-1}(-mgl) & 0 & K'^{-1}(-c) \end{bmatrix} \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ K'^{-1}au \\ 0 \end{bmatrix} \end{aligned} \quad (2.19)$$

となる。ただし、 K は、

$$K = \begin{bmatrix} M+m & -ml \\ -ml & J+ml^2 \end{bmatrix}$$

である。

今後、(2.16) 式を鉛直上向き基準の線形状態方程式とし、(2.19) 式を鉛直下向き基準の線形状態方程式とする。

2.1.2 観測方程式

2つの観測出力は

$$y_1 = c_1 r$$

$$y_2 = c_2 \theta$$

のように表される。ここで、 c_1 は変位・電圧変換係数、 c_2 は角度・電圧変換係数である。これから成るベクトル出力 y を

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

のように定義すると、倒立振子系に対する観測方程式として

$$y = Cx$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 \end{bmatrix} \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix} \quad (2.20)$$

を得ることができる。

なお、鉛直上向きを基準とした場合でも鉛直下向きを基準とした場合でも出力方程式は変わらない。

2.2 倒立振子のパラメータの同定

数式モデル (2.15)、(2.1.1)、(2.20) に含まれる物理パラメータを実際の倒立振子系で実験を行い同定する。

2.2.1 m と l の測定

倒立振子系から振子を取り外し、バネ秤で振子の質量 m を測定する。つぎに、振子を鋼尺のエッジ上でバランスさせて、重心の位置を定め、 l を測定する。以下に測定した結果を示す。

$$m = 0.031[\text{kg}]$$

$$l = 0.15[\text{m}]$$

2.2.2 a の測定

モータに一定電圧を加え、ばねばかりで台車を引き、台車が正の方向に動き出すときの力 ($au + \text{摩擦力}$) を f_{max} 、負の方向に動き出すときの力 ($au - \text{摩擦力}$) を f_{min} とする。図 2.2 に示すように u と f_{max}, f_{min} の関係をいくつかの電圧について調べ、最小 2 乗法によって 1 次関数を求め、この傾きを a とする。[2]

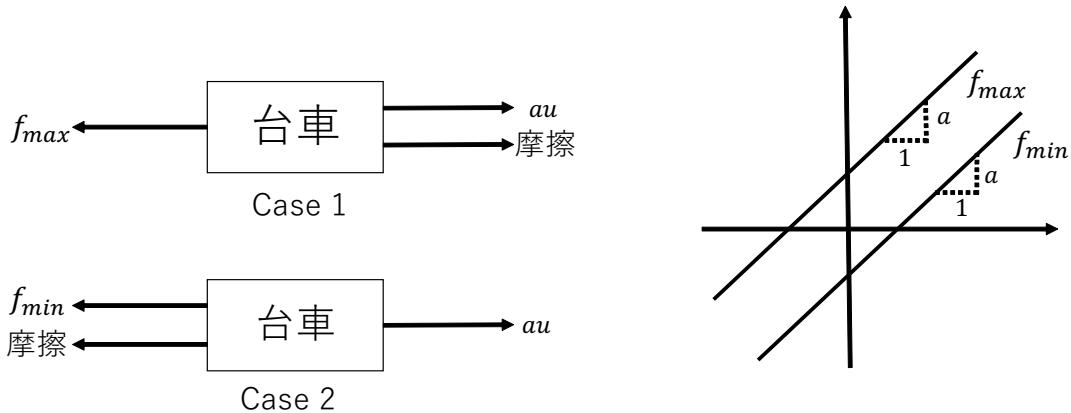


図 2.2: パラメータ a の決定

なお、振子は台車から取り外して測定を行う。以下に測定した結果から得られた一次関数の傾き a を示す。

$$a = 0.062[\text{Kg/V}] = 0.61[\text{N/V}]$$

2.2.3 Jとcの測定

振子を自由振動させることにより、 J と c を測定できる。その数式モデルは鉛直下向きを基準として

$$(J + ml^2)\ddot{\theta} - mgl \sin \theta + c\dot{\theta} = 0 \quad (2.21)$$

$$y_2 = c_2\theta \quad (2.22)$$

で与えられる。 θ を微小範囲で考えると、(2.21),(2.22)式は

$$\ddot{y}_2 + 2\zeta\omega_n\dot{y}_2 + \omega_n^2y_2 = 0$$

ただし、

$$\zeta = \frac{c}{2\sqrt{mgl(J+ml^2)}}, \quad \omega_n = \sqrt{\frac{mgl}{J+ml^2}}$$

のように書くことができる。この解は

$$0 < \zeta < 1$$

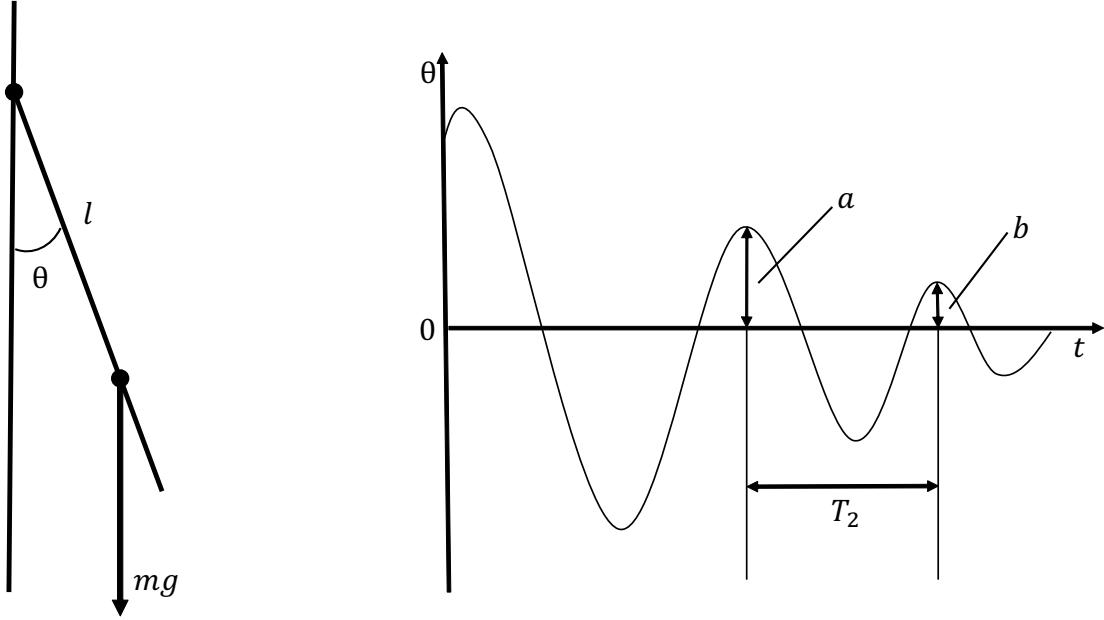
のとき、減衰振動となり

$$y_2(t) = \frac{y_2(0)}{\sqrt{1-\zeta^2}} \exp(-\omega_n\zeta t) \sin(\omega_n\sqrt{1-\zeta^2}t + \phi)$$

ただし

$$\phi = \tan^{-1} \frac{\sqrt{1-\zeta^2}}{\zeta}$$

で与えられる。[2]

図 2.3: J と c の測定

いま、減衰振動の周期を T_2 とし、時刻 t_1 と時刻 $t_2 = t_1 + T_2$ において波形 $y_2(t)$ の山が隣合うものとする。このときの振幅の減衰比は

$$\frac{|y_2(t_2)|}{|y_2(t_1)|} = \exp(-\lambda)$$

ただし

$$\lambda = \frac{2\pi\zeta}{\omega_n \sqrt{1 - \zeta^2}}$$

となる。この λ は対数減衰比と呼ばれる。また

$$T_2 = \frac{2\pi}{\omega_n \sqrt{1 - \zeta^2}}$$

が成り立つ。したがって、 J と c は

$$J = \frac{mglT_2^2}{4\pi^2 + \lambda^2} - ml^2, \quad c = \frac{2\lambda(J + ml^2)}{T_2}$$

のように与えられる。[2] 以下に振子を自由振動させ得られたデータから計算した J と c を示す。

$$J = 2.5 \times 10^{-4} [\text{kNm}^2]$$

$$c = 5.4 \times 10^{-5} [\text{kgm}^2/\text{s}]$$

2.2.4 M と f の測定

M と f の測定方法には二通りがある。

ステップ応答による測定法

ここでは、台車をアンプ・モータブーリ・ベルト・台車系の等価質量と等価摩擦係数とし、台車のステップ応答を測定することで M と f を決定する。ただし、振り子は台車から取り外した状態で測定を行う。このときの運動方程式は

$$M\ddot{r} = au - f\dot{r}$$

であり、 u から r までの伝達関数 G は

$$G(s) = \frac{K}{s(Ts + 1)}$$

となる。ただし、

$$K = \frac{a}{f}, \quad T = \frac{M}{f} \quad (2.23)$$

である。初期状態を 0 とするとき、このシステムのステップ応答は

$$r(t) = KU_0 \left(Te^{\frac{-t}{T}} + t - T \right) \quad (2.24)$$

である。

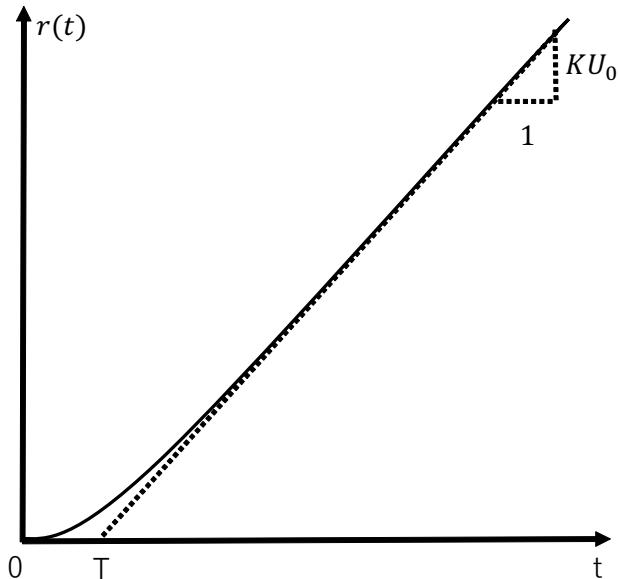


図 2.4: 台車のステップ応答

ただし、 U_0 はステップの高さである。 (2.24) において $t \rightarrow \infty$ とすれば

$$r(t) = KU_0(t - T)$$

となり、図 2.4 を参考に T と K をもとめ、 (2.23) 式より M と f を決定することができる。以下にこの方法を用いて同定したパラメータを示す。

$$M = 6.9E - 1[\text{kg}]$$

$$f = 7.6[\text{kg/s}]$$

フィードバック入力による測定法

ここでは、入力にステップ応答ではなく、以下に示すようなフィードバック入力を加える。

$$u = k_c(y_c - y) \quad (2.25)$$

ただし、 y_c は目標値、 y は出力、 k_c はフィードバックゲインである。本節では $k_c = 2500$ とする。このようにすることで不足制動の 2 次系 ($\zeta < 1$) を実現させる。

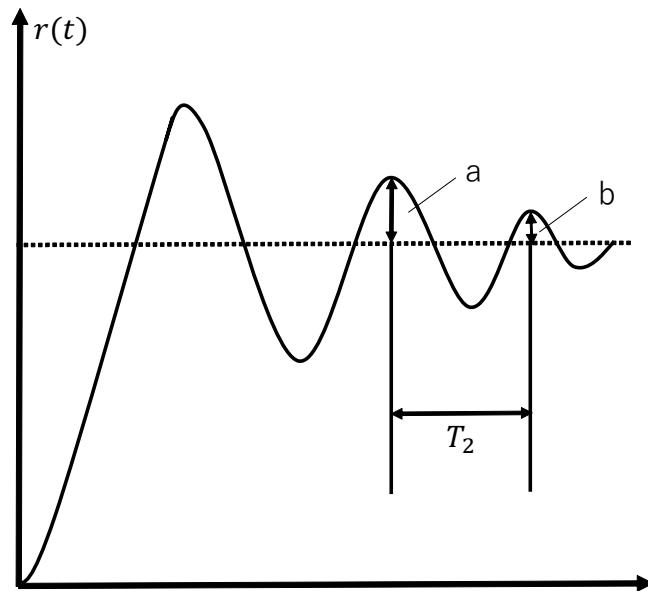


図 2.5: 台車のフィードバック応答

図 2.5 を参考にして M と f を同定する。この方法は J と c の同定の際に用いた方法とほぼ同じであるため、詳しい説明はそちらに譲る。

台車のフィードバック応答から求めた λ と T_2 から以下の式を用いて ζ と ω_n を計算し求める。

$$\lambda = \frac{2\pi\zeta}{\sqrt{1-\zeta^2}} \quad (2.26)$$

$$T_2 = \frac{2\pi}{\omega_n \sqrt{1-\zeta^2}} \quad (2.27)$$

上の式を式変形して ζ と ω_n イコールの式にすると

$$\zeta = \frac{\lambda}{\sqrt{4\pi^2 + \lambda^2}} \quad (2.28)$$

$$\omega_n = \frac{2\pi}{T_2 \sqrt{1-\zeta^2}} \quad (2.29)$$

以上の式から求めた ζ と ω_n は以下の二次系の伝達関数の基本形に代入することで同定に用いたフィードバック制御系の伝達関数を求めることができる

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.30)$$

また、今回のフィードバック制御系におけるブロック線図は以下のようにになる。

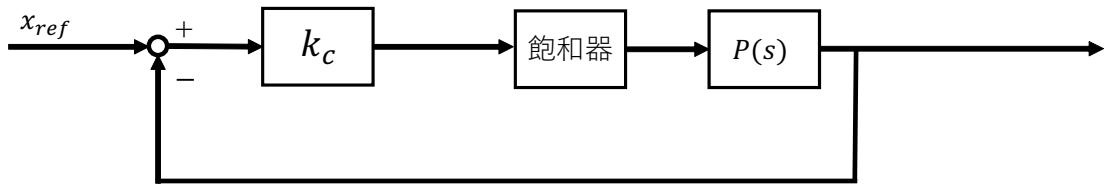


図 2.6: フィードバック制御系のブロック線図

ここで、 $P(s)$ は台車の伝達関数であり以下の式で表される。

$$P(s) = \frac{K}{s(Ts + 1)} \quad (2.31)$$

ただし、 T は M/f 、 K は a/f である。上のブロック線図から伝達関数を求める。しかし、飽和システムを含んでいると伝達関数を求めることがないので、ここでは飽和システムがなくても台車は問題なく動作するものとして仮定する。以上の仮定から伝達関数は

$$G(s) = \frac{AK/T}{s^2 + (1/T)s + (AK/T)} \quad (2.32)$$

となる。ただし A はゲインである。 (2.31) 式と (2.32) 式を係数比較し、 M, f イコールの式にすると以下のようになる。

$$M = \frac{aA}{\omega_n^2}, \quad f = 2\zeta\omega_n^2 M \quad (2.33)$$

よって、(2.33) 式から M と f は以下のように求まる。

$$M = 1.59[\text{kg}]$$

$$f = 13.2[\text{kg}/\text{s}]$$

M と f の決定

パラメータ M と f についてはステップ応答による方法とフィードバックによる方法の2通りから同定を行った。しかし、それぞれの方法で求めたパラメータを比較するとまるで違うことがわかる。これはフィードバックによる方法が十分に正確な方法ではなかったためといえる。その原因は、飽和システムを含めずに同定を行ったことではないかと考える。シミュレーションにおいては入力にどのような値を加えても倒立振子系に何ら影響はないが実際の倒立振子においてはその入力できる値には制限がある。この制限を考えるか否かで結果が変わってくるはずである。

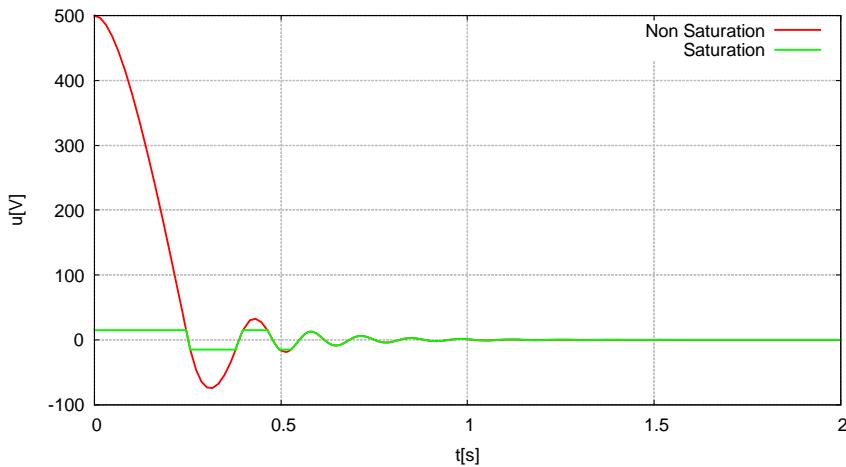


図 2.7: 飽和器の有無によるフィードバック入力の比較

図 2.7 は図 2.6 のブロック線図において飽和器を含めた場合の入力と飽和器を含めなかつた場合の入力を描画させたものである。この図より、飽和器がない (non Saturation) 場合、入力は最大 500(V) まで加えられることがわかる。本実験で用いる倒立振子系の入力限界は $-15 < u < 15$ であるので、飽和器を加えた場合、その範囲を超える入力は絶対値 15(V) に制限されている。つまり、フィードバックによる方法で同定したパラメータは飽和器を考慮していないので、現実の倒立振子系とは違うパラメータであるといえる。また、今回はフィードバックゲインを 2500 としたが、フィードバックゲインを小さくすると、入力もそれに伴い小さくなるので、現実の倒立振子系に近いパラメータを同定することができるはずである。以上から本実験においてフィードバックを行う場合、入力が大きく跳ね上がる可能性があるので、必ず飽和システムが必要であるといえる。なので、飽和システム

が存在する限り伝達関数を求めることができないので、正確なパラメータの計算を行うことができないといえる。以降、パラメータ M, f はステップ応答による方法で同定した値を用いる。

2.2.5 c_1 と c_2 の測定

c_1 と c_2 に関しては

$$c_1 = 1.0[\text{V}/\text{m}]$$

$$c_2 = 1.0[\text{V}/\text{rad}]$$

というようにソフトウェアに設定してあるものを用いた。

2.2.6 同定結果

同定実験において同定したパラメータを表にまとめる。

表 2.1: 同定したパラメータの一覧

パラメータ	同定した値
$m[\text{kg}]$	0.031
$l[\text{m}]$	0.15
$M[\text{kg}]$	0.69
$f[\text{kg}/\text{s}]$	7.6
$J[\text{kNm}^2]$	2.5×10^{-4}
$c[\text{kNm}^2/\text{s}]$	5.4×10^{-5}
$a[\text{N}/\text{V}]$	0.61
$c_1[\text{V}/\text{m}]$	1.0
$c_2[\text{V}/\text{rad}]$	1.0

2.3 パラメータの検証

前節では、実験によって倒立振子のパラメータを同定した。しかし、その同定したパラメータがどれほど有効性があるか現時点では全く分からぬ状況である。そこで、本節では同定したパラメータの有効性をがどれほどなのかシミュレーションを用いて検証を行う。シミュレーションに用いたツールは JAMOX である。ただし、直接同定を行った m, l や最初から設定してあった c_1, c_2 については検証は行わない。

2.3.1 振子の自由振動シミュレーション

ここでは、 J, c の検証を行う。以下にシミュレーションと実験データを描画したグラフを示す。

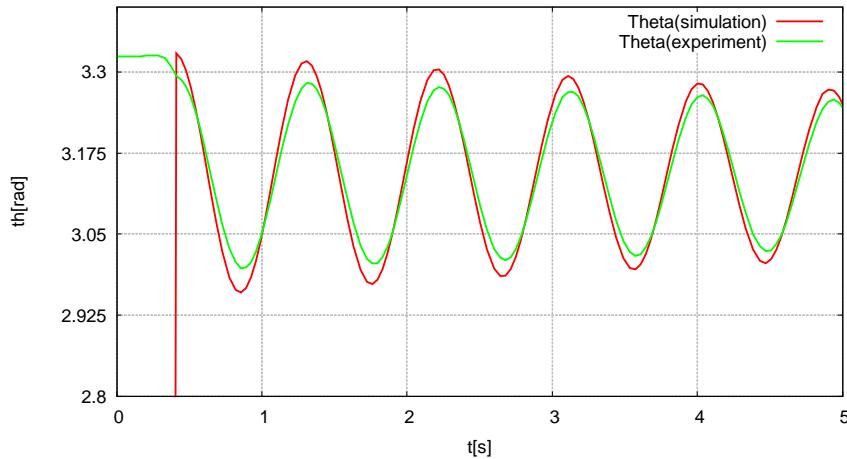


図 2.8: 振子の自由振動シミュレーション

図 2.8において、シミュレーションは 0.32(s) 程遅らせて実行しているので、それ以前の時間の角度は 0°である。自由振動の初期角度は 180°を基準として 10°である。図 2.8 よりシミュレーションと実験結果に少し差異はあるが大きな違いはないといえる。よって同定したパラメータ J, c は有効な値であるといえる。

2.3.2 台車のステップ応答による方法のシミュレーション

ここでは、 M, f, a の検証を行う。 M, f はステップ応答による方法で求めた値を用いる。以下にシミュレーションと実験データを描画したグラフを示す。

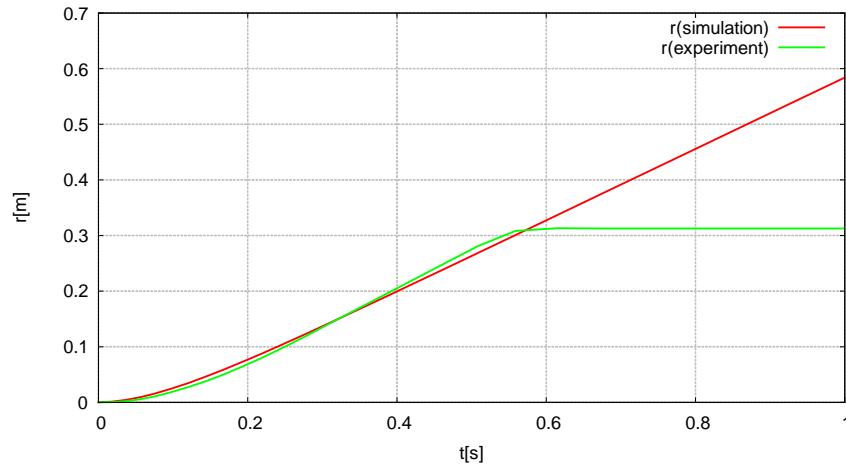


図 2.9: 台車のステップ応答シミュレーションと実験結果の比較

図 (2.9) よりシミュレーションと実験結果に少し差異はあるが大きな違いはないといえる。0.6 秒以降のグラフが大幅に異なっているが、これはシミュレーションの場合は台車の可動範囲に制限がないためずっと台車が移動し続けるが、実験の場合は台車の可動範囲に制限があるため、ある時間を境に移動が止まっている。また、このときの入力電圧は 8[V] である。以上から同定したパラメータ M, f, a は有効な値といえる。

2.3.3 台車のフィードバック応答シミュレーション

ここでは、 M, f, a の検証を行う。 M, f はフィードバックによる方法で求めた値を用いる。以下にシミュレーションと実験データを描画したグラフを示す。ただし、フィードバックゲインは 1500 である。

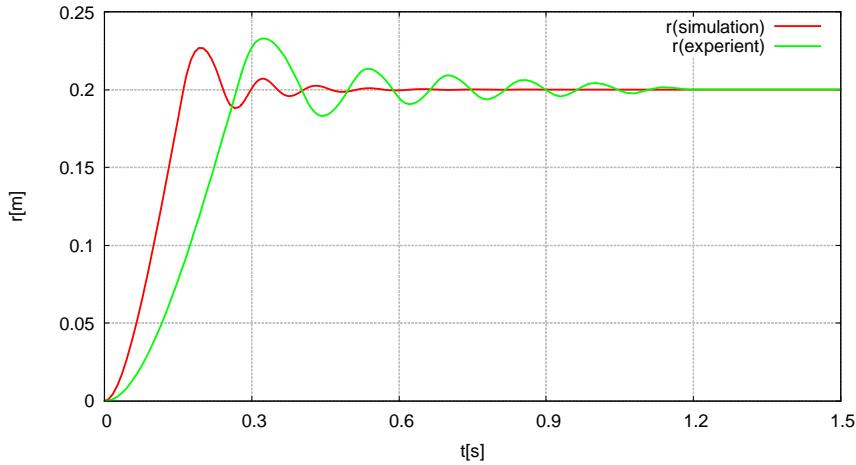


図 2.10: フィードバックによる方法のシミュレーションと実験結果の比較

図 2.10 よりシミュレーションと実験結果には大きな差異があることが確認できる。上述した通りこの方法で同定したパラメータは現実の倒立振子系に即しておらず、結果実験で得られた波形との比較において大きな違いが出てきてしまう。よってこの方法で同定したパラメータ M, f には有効性がないといえる。

2.4 設計（線形）モデルの決定

ここまでで、パラメータを同定し、その有効性についても確かめた。よって、同定すべきパラメータを決定できたのでシステム行列 A、入力行列 B、出力行列 C を MATX を用いて計算し、倒立振子の線形モデルを確定する。MATX で計算した各行列を以下に示す。

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0.32 & -10.9 & 0.00038 \\ 0 & 50 & 53 & -0.059 \end{bmatrix} \quad (2.34)$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0.87 \\ -4.3 \end{bmatrix} \quad (2.35)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (2.36)$$

ただし、倒立振子を立たせることを目的とするので、上向きを基準としたときの線形状態方程式である。

第3章 制御系設計

3.1 システム解析

前章で確定した線形モデルについてシステム解析を行う。これにより今回用いるモデル(上向きを基準とした線形状態方程式)が安定化制御可能か判定することができる。具体的には可制御性と可観測性を調べ、それらが存在すれば安定化制御可能であるといえる。以下で行う計算はすべて MATLAB を用いた。

3.1.1 安定性

システムの極 (A の固有値) を計算した結果 D を以下に示す。

$$D = \begin{bmatrix} 7.0 \\ 0 \\ -6.8 \\ -11 \end{bmatrix} \quad (3.1)$$

(4.1) 式より 1 行目が不安定であり、2 行目が安定限界であるので、今回用いるモデルは不安定であるといえる。

3.1.2 可制御性

可制御性行列は以下のようになる。

$$N_c = \begin{bmatrix} C & CA & CA^2 & CA^3 \end{bmatrix}$$

上の行列よりランクは 4 になれば可制御性があるといえる。ランクを計算したところ

$$\text{rank} = 4$$

となった。よって可制御性を確認できる。

3.1.3 可制御性

可観測性行列は以下のようになる。

$$N_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix}$$

上の行列よりランクは 4 になれば可観測性があるといえる。ランクを計算したところ

$$\text{rank} = 4$$

となった。よって、可観測性を確認できる。

以上から倒立振子系の上向き基準の線形モデルは不安定なシステムであるが、4つの状態を観測することができ、制御することが可能なシステムといえる。次の節では倒立振子を立たせるための制御器を設計していく。その際に前章で計算したシステム行列 A と入力行列 B を用いる。

3.2 状態フィードバックの設計

制御器を設計していく第一段階として、状態フィードバック F の設計を行う。 F は、システムを安定化する状態フィードバック

$$u = -Fx$$

として求めればよい。また、本実験においては台車が目標値へ移動を行うので、目標値 x_{ref} として以下を設計することになる。

$$u(t) = F(x_{ref} - x) \quad (3.2)$$

この状態フィードバックの設計法には、極配置に基づく状態フィードバック法と LQ 最適制御に基づく状態フィードバック法の 2 通りがあるが今回は後者の方法を用いて設計を行う。

さて、(3.2) 式を LQ 問題の解として得るために、2 次形式評価関数

$$J = \int_0^\infty (x^T Q x + R u^2) dt \quad (3.3)$$

$$Q = \text{diag}(q_1^2, q_2^2, q_3^2, q_4^2), \quad R = 1 \quad (3.4)$$

を考える。ただし、 $\text{diag}(\dots)$ は、対角行列を表す。これは

$$J = \int_0^\infty (q_1^2 r^2 + q_2^2 \theta^2 + q_3^2 \dot{r}^2 + q_4^2 \dot{\theta}^2) dt \quad (3.5)$$

のように表されることから、 q_1, q_2, q_3, q_4 は台車位置 r 、振り子角度 θ 、台車速度 \dot{r} 、振子角速度 $\dot{\theta}$ の間のバランスをとる重み係数である。

(3.3) と (3.4) 式を最小にする (3.2) 式における F は、リカッチの方程式

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

の解 $P > 0$ を求めて

$$F = R^{-1}B^T P$$

のよう与えられる。

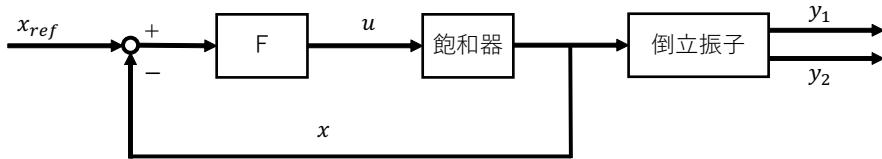


図 3.1: 状態フィードバックを含めたブロック線図

3.3 最小次元オブザーバの設計

第二段階として、

$$\hat{x} \rightarrow x \quad (t \rightarrow \infty) \quad (3.6)$$

を満足させる状態を推定する状態観測器（最小次元オブザーバ）

$$\dot{\hat{z}}(t) = \hat{A}\hat{z}(t) + \hat{B}y(t) + \hat{J}u(t) \quad (3.7)$$

$$\hat{x}(t) = \hat{C}\hat{z}(t) + \hat{D}y(t) \quad (3.8)$$

をゴピナスの方法で設計する。この方法は、ある行列 $U(1 \times 4)$ が存在して

$$\begin{aligned} UA &= \hat{A}U + \hat{B}C \\ UB &= \hat{J}I = \hat{C}U + \hat{D}C \end{aligned}$$

かつ \hat{A} が安定行列であることを満足する方法であり、オブザーバ (3.7)、(3.8) が (3.6) 式を満足するための十分条件でもある。なお、本実験で用いる倒立振子系は r, θ はセンサーを用いて計測できるが、 $\dot{r}, \dot{\theta}$ においては計測するためのセンサーが存在しないため、推測でしかこれらを得ることができない。その推測を行うのに、最小次元オブザーバーを用いる。

具体的には、オブザーバの係数行列 $\hat{A}, \hat{B}, \hat{J}, \hat{C}, \hat{D}$ を求める。これらは適当なオブザーバの極を選択することで得ることができる。。また、オブザーバの極を決める際に、状態フィードバック制御

$$u(t) = F(x_{ref}(t) - x(t))$$

による閉ループ系

$$\dot{x}(t) = (A - BF)x(t) + BFx_{ref}(t)$$

の極との位置関係を考慮する必要がある。

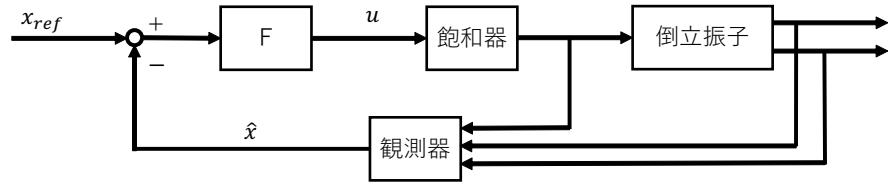


図 3.2: 最小次元オブザーバーを含めたブロック線図

3.4 コントローラの離散化

コントローラは連続時間で記述された (3.2)、(3.7)、(3.8) 式で与えられるが、計算機制御のためにはこれらを離散時間で記述しなければならない。これらを離散化したものと離散時間コントローラと呼ぶ。離散時間コントローラはサンプリング周期を Δ とすると、以下の式で与えられる。

$$z[k+1] = \hat{A}_d z[k] + \hat{B}_d y[k] + \hat{J}_d u[k] \quad (3.9)$$

$$\hat{x}[k] = \hat{C}_d z[k] + \hat{D}_d y[k] \quad (3.10)$$

$$u[k] = F(x_{ref}[k] - \hat{x}[k]) \quad (3.11)$$

ただし、 $k = 0, 1, \dots$ であり

$$\begin{pmatrix} \hat{A}_d & [\hat{B}_d \ \hat{J}_d] \\ 0_{3 \times 2} & I_3 \end{pmatrix} = \exp\left(\Delta \begin{pmatrix} \hat{A} & [\hat{B} \ \hat{J}] \\ 0_{3 \times 2} & I_3 \end{pmatrix}\right)$$

である。具体的には、適当なサンプリング周期を Δ を設定すればよい。

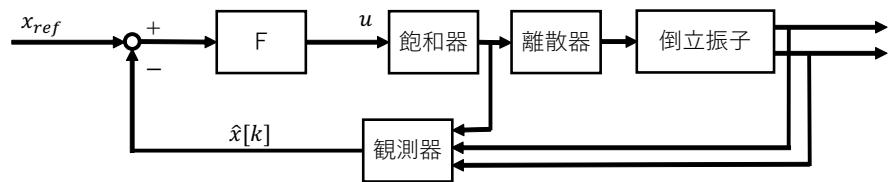


図 3.3: 離散器（0次ホールド）を含めたブロック線図

3.5 振り上げ制御及び安定化の実現

ここでは振子の振り上げ制御を行うためのコントローラを設計する。振り上げ制御とは、振り子を下向きにしたままで台車を動かすことで振り子を倒立させようという制御のことである。この制御には振り子を振り上げる制御と振り子の安定化制御を使い分けることで実現させる。具体的には下向きの振り子を振り上げ制御により徐々に上に持っていく（ θ を0に近づけていく）、このときある一定の角度を境界値と定め、その境界値まで振り子の角度がちいさくなったらところで制御を安定化制御に切り替え、安定化制御を行うというものである。振り子の安定化制御についてはこれまでのコントローラを用いるので、ここでは振り子の振り上げ制御の理論とその実現方法について述べる。

台車と振り子の運動方程式は

$$(M+m)\ddot{r} + ml \cos \theta \ddot{\theta} = -f\dot{r} + ml \sin \theta \dot{\theta}^2 + au \quad (3.12)$$

$$ml \cos \theta \ddot{r} + (J+ml^2)\ddot{\theta} = mgl \sin \theta - c\dot{\theta} \quad (3.13)$$

で与えられる。振り子が垂直上向きのときを基準とする振り子の力学的エネルギーは

$$E = \frac{1}{2}(J+ml^2)\dot{\theta}^2 + mgl(\cos \theta - 1)$$

で与えられる。第一項が回転に関するエネルギーであり、第二項が傾きを考慮した位置エネルギーである。なお、基準において静止しているとき、力学的エネルギーは $E=0$ である。このとき、力学的エネルギーの時間微分は

$$\frac{dE}{dt} = (J+ml)\ddot{\theta}\dot{\theta} - mgl\dot{\theta} \sin \theta \quad (3.14)$$

となる。振り上げ制御のために、次の制御則を用いる。

$$u = \frac{1}{a} \left(f\dot{r} - ml \sin \theta \dot{\theta}^2 + ml \cos \theta \ddot{\theta} + (M+m)v \right) \quad (3.15)$$

$$v = -\frac{c\dot{\theta}}{ml \cos \theta} + k(E - E_0) \text{sign}(\dot{\theta} \cos \theta) \quad (3.16)$$

ただし、 sign は符号関数であり、引数の値が負のとき -1 、正のとき 1 、0のとき 0 となる。 (3.15) 式を (3.12) 式に代入すると、

$$\ddot{r} = v \quad (3.17)$$

を得る。 (3.17) 式と (3.16) 式を (3.13) 式に代入すると

$$(J+ml^2)\ddot{\theta} = mgl \sin \theta - ml \cos \theta (k(E - E_0) \text{sign}(\dot{\theta} \cos \theta))$$

を得る。この式を (3.14) 式に代入すると

$$\begin{aligned} \frac{dE}{dt} &= -ml\dot{\theta}(k(E - E_0) \text{sign}(\dot{\theta} \cos \theta)) \\ &= -mlk(E - E_0) \text{sign}(\dot{\theta} \cos \theta)(\dot{\theta} \cos \theta) \end{aligned}$$

となる。リアプノフ関数の候補として、

$$V = \frac{(E - E_0)^2}{2} \geq 0$$

を考える。V の時間微分を求めるとき、

$$\begin{aligned} \frac{dV}{dt} &= (E - E_0) \frac{dE}{dt} \\ &= -mlk(E - E_0)^2 \text{sign}(\dot{\theta} \cos \theta)(\dot{\theta} \cos \theta) \leq 0 \end{aligned}$$

これより、 $\dot{\theta} \cos \theta \neq 0$ のとき、 $\dot{V} < 0$ であるので V は減少して 0 に収束し、E は E_0 に収束する。なお、k を大きくすると、早く E が E_0 に収束する。実際の制御では、台車の加速度目標 v を制限し、

$$\begin{aligned} u &= \frac{1}{a} (f\dot{r} - ml \sin \theta \dot{\theta}^2 + ml \cos \theta \ddot{\theta} + (M+m)v) \\ v &= -\frac{c\dot{\theta}}{ml \cos \theta} + \text{sat}_{ng}(k(E - E_0)\text{sign}(\dot{\theta} \cos \theta)) \end{aligned}$$

とする。ただし、sat は最小値が $-ng$ 、最大値が ng の飽和関数である。n は、重力加速度（鉛直下向き）と台車の加速度（水平方向）の比である。

具体的には、k と重力加速度と台車の加速度の比である n を調整し、振り上げ制御を実現させる。

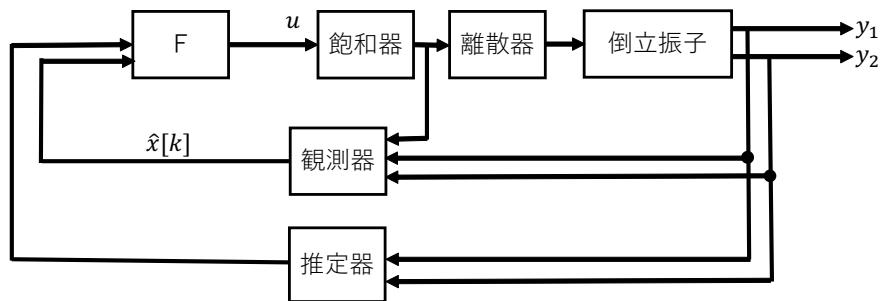


図 3.4: 振り上げ制御及び安定化制御を行うためのブロック線図

第4章 シミュレーション

目標値変更のシミュレーションによって制御器に関する各パラメータの有効な値を決定する。シミュレーションには JAMOX を用いた。

4.1 重み行列の変更による制御性能評価

LQ 最適制御に基づく状態フィードバック F を設計するために連続時間線形二次レギュレーターを用いる。この時、連続時間線形二次レギュレーターにはシステム行列 A と入力行列 B と重み行列 Q, R が必要である。重み行列は、(3.2) 式で与え、この重み行列を調整することで倒立振子の安定化制御の性能を高めることができる。そこで、シミュレーションを用いて重み行列を変更させたときの制御性能について考察していく、どのような重み行列を用いればよいか決定する。

今回は行列 R の値は変更しないので、行列 Q のみ値を変更してシミュレーション結果を考察する。便宜上、重み行列 Q の対角成分を左から第 1 成分、第 2 成分、第 3 成分、第 4 成分と呼ぶことにする。以下にシミュレーションを行う各パターンのパラメータをまとめた表とその時のシミュレーション結果を示す。

表 4.1: 重み行列の変更パターン

	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
パターン 1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
パターン 2	$Q_2:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
パターン 3	$Q_3:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$

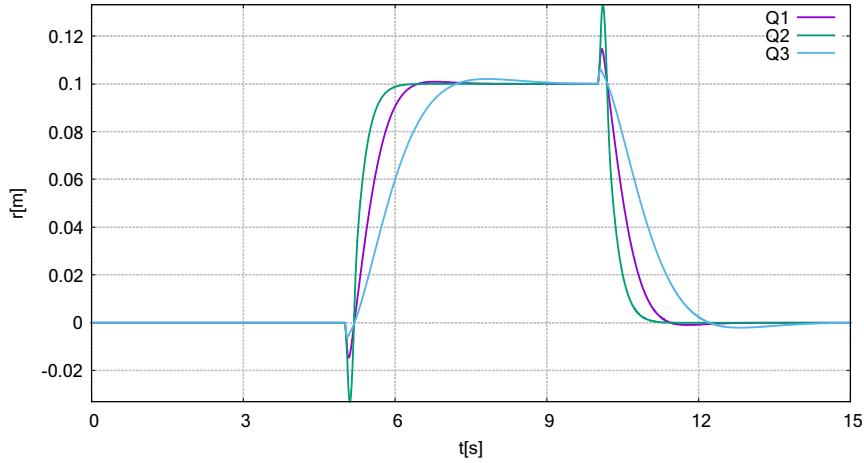


図 4.1: 重み行列での比較結果（台車位置）

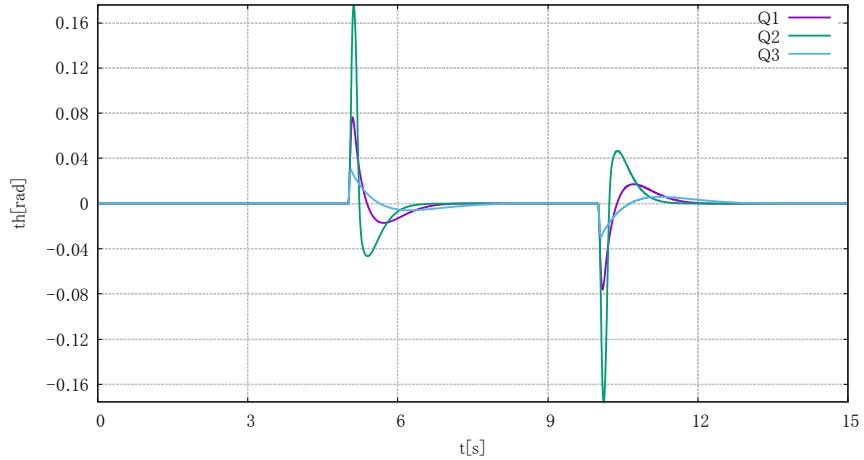


図 4.2: 重み行列での比較結果（振子角度）

図(4.1)より Q_2 の応答が一番早く、 Q_3 の応答が一番遅いことが確認できる。このことから台車の位置 r の応答をよくするには重み行列 Q の第1成分を大きくすればよいとわかる。また、 Q_1 と Q_3 の第1成分は同じであるのにもかかわらず、 Q_1 の方が応答が早いことが確認できる。これは振子の角度 θ の応答をよくする第2成分を大きくした分、バランスが第2成分のほうに偏ってしまい、台車の位置の応答が悪くなつたといえる。

図(4.2)より、振子の角度についても同じことが言え、第2成分を大きくしている Q_3 の応答が一番よく、 Q_2 の応答が一番悪い結果となっている。

以上から重み行列の調整については以下のことが言える。

- 重み行列 Q の各成分は $r, \theta, \dot{r}, \dot{\theta}$ に対応している

- 応答をよくしたい状態があれば、その状態に対応する成分の値を大きくすればよい
- その場合、大きくした成分に対応する状態にバランスが偏る(つまり、ほかの状態の応答が悪くなる)

これらを考慮して重み行列を決定すれよい。

4.2 オブザーバの極の変更による制御性能評価

前章で述べたように、最小次元オブザーバを設計するためにゴビナスの方法を用いる。この時、システム行列 A 、入力行列 B 、出力行列 C 、オブザーバの極 P が必要である。オブザーバの極 P を調整することで倒立振子の安定化制御の性能を高めることができる。同様にシミュレーションによって制御性能を考察していき、適当なオブザーバの極 P を決定する。

1つ考慮しなければならないのは、オブザーバの極と閉ループ系の極(つまり、 $(A - BF)$ の固有値)との位置関係である。閉ループ系の極のうち虚軸に最も近い極を λ_{max} としたとき、オブザーバーの極 P は

$$\text{Re}(P) < 5\text{Re}(\lambda_{max})$$

を考慮して設定する必要がある。表(4.1)の Q_2 を用いて設計した状態フィードバック F における閉ループ系の極は

$$D = \begin{bmatrix} -0.08 \\ -6.3 + 1.3i \\ -6.3 - 1.3i \\ -13 \end{bmatrix} \quad (4.1)$$

である。この中で一番虚軸に近い極は -0.08 である。よって、少なくともオブザーバーの極 P は -0.4 より小さい値をとればよいとわかる。

以下にシミュレーションを行う各パターンのパラメータをまとめた表とその時のシミュレーション結果を示す。

表 4.2: オブザーバの極の変更パターン

	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
パターン 1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
パターン 2	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_2:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$

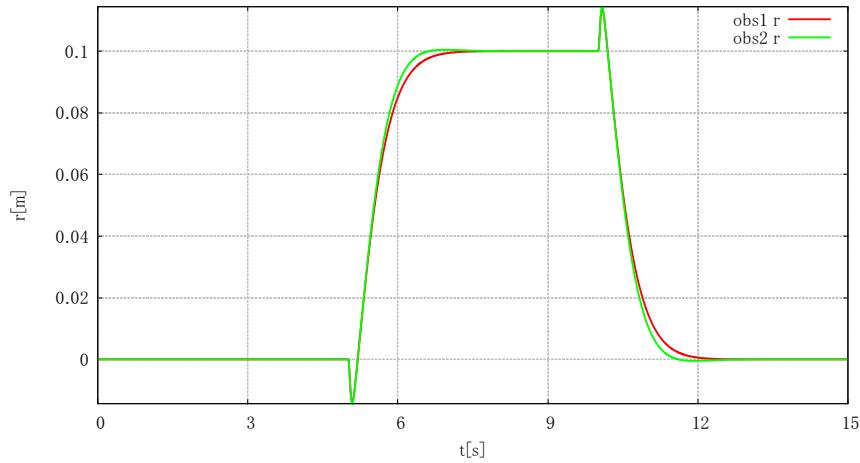


図 4.3: オブザーバーの極での比較結果（台車位置）

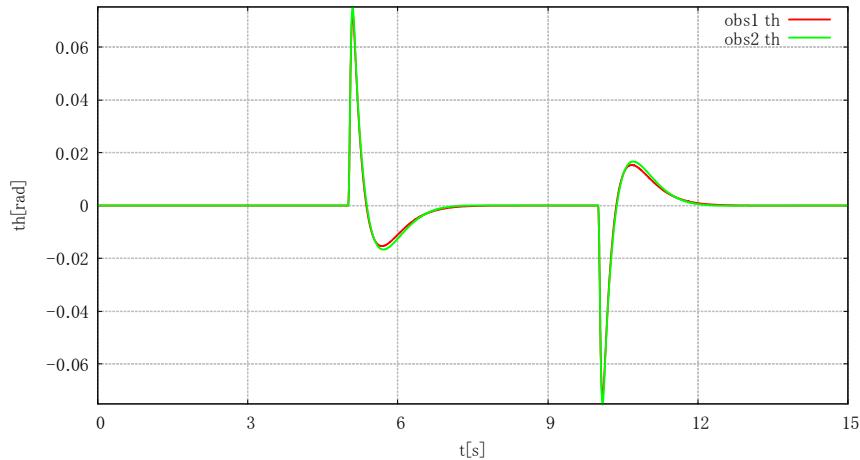


図 4.4: オブザーバーの極での比較結果（振子角度）

図 4.3 と図 4.4 よりオブザーバの極の違いによる大きな違いは確認できない。ここで、倒立振子の状態 x (台車の速度と振り子の角速度) とオブザーバで推定した値 \hat{x} (台車の速度と振り子の角速度) との差である推定誤差 ($x - \hat{x}$) の応答波形からオブザーバの極の違いによる考察をする。以下に台車の速度と振り子の角速度の推定誤差の図を示す。

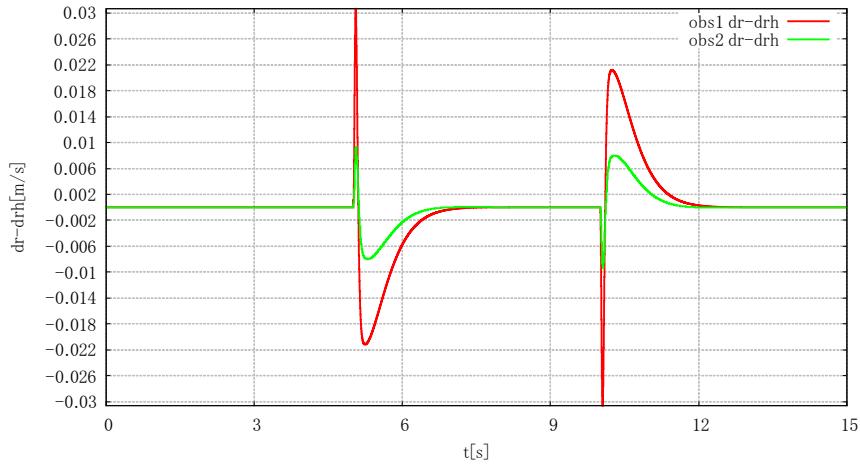


図 4.5: オブザーバーの極での推定誤差の比較（台車の速度）

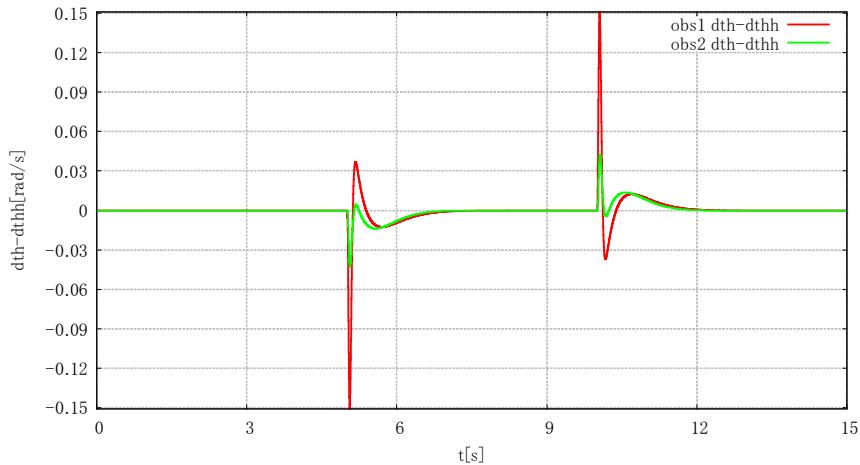


図 4.6: オブザーバーの極での推定誤差の比較（振子の角速度）

図 4.5 と図 4.6 よりオブザーバの極が虚軸から遠いパターン 2 のほうが推定誤差の収束が若干遅いことがわかる。また、その大きさもパターン 2 のほうが大きいこともわかる。

以上からオブザーバの極配置の調整については以下のことが言える。

- オブザーバの極は虚軸に近いほうが推定誤差が小さくなり応答はよくなるといえる。
- ただし、閉ループ系の極との位置関係を考慮するひつようがある。

これらを満たすオブザーバの極を決定すればよい。

4.3 サンプリング周期の変更による制御性能評価

離散化を行うサンプリング周期 Δ を変更することで倒立振子の安定化制御の性能を高めることができる。同様にシミュレーションによって制御性能を考察していく、どのようなサンプリング周期をとればよいか決定する。ただし、サンプリング周期が短すぎると実験で用いる倒立振子系においてシステム全体がハングアップする恐れがあるため、そのような値においてはシミュレーションを行わない。

以下にシミュレーションを行う各パターンのパラメータをまとめた表とその時のシミュレーション結果を示す。

表 4.3: サンプリング周期の変更パターン

	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
パターン 1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
パターン 2	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_2:0.01$

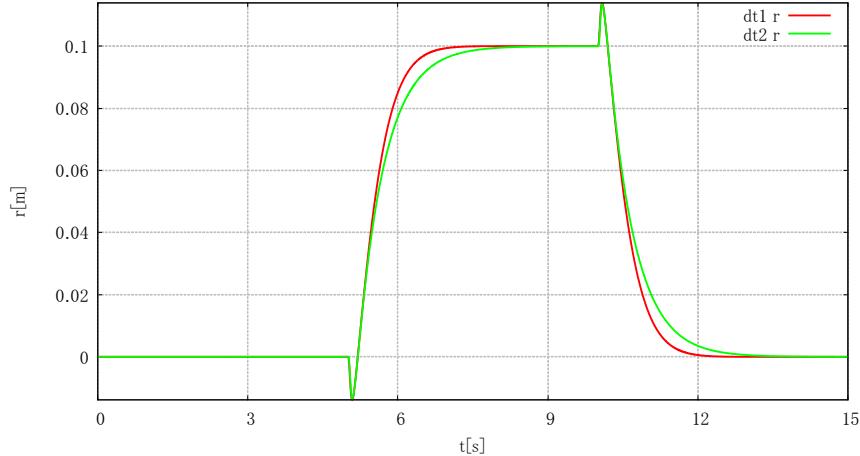


図 4.7: サンプリング周期での比較結果（台車位置）

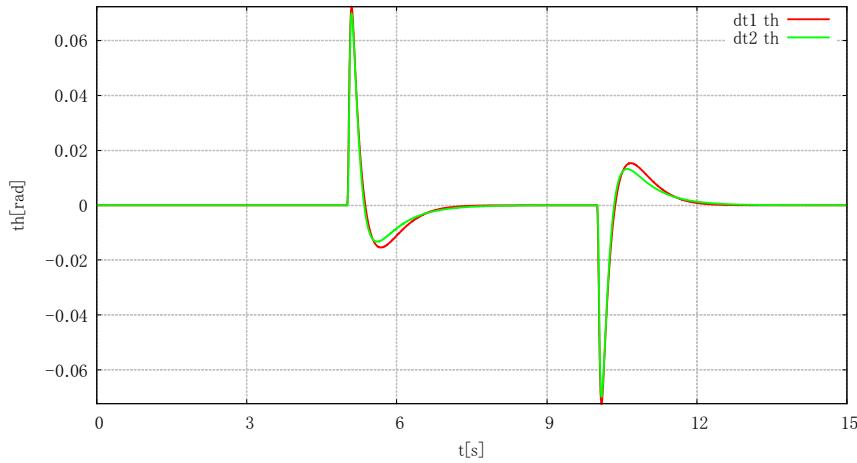


図 4.8: サンプリング周期での比較結果（振子角度）

図 4.7 から、パターン 1 のほうが若干応答が早いといえる。しかし、図 4.7 と図 4.8 からは大きな違いを確認することはできない。先ほどと同様に倒立振子の状態 $x(t)$ とオブザーバで推定した値 $\hat{x}[k \cdot T]$ との差である推定誤差 ($x(t) - \hat{x}[k \cdot T]$) の応答波形からサンプリング周期の違いによる考察をする。以下に台車の速度と振り子の角速度の推定誤差の図を示す。

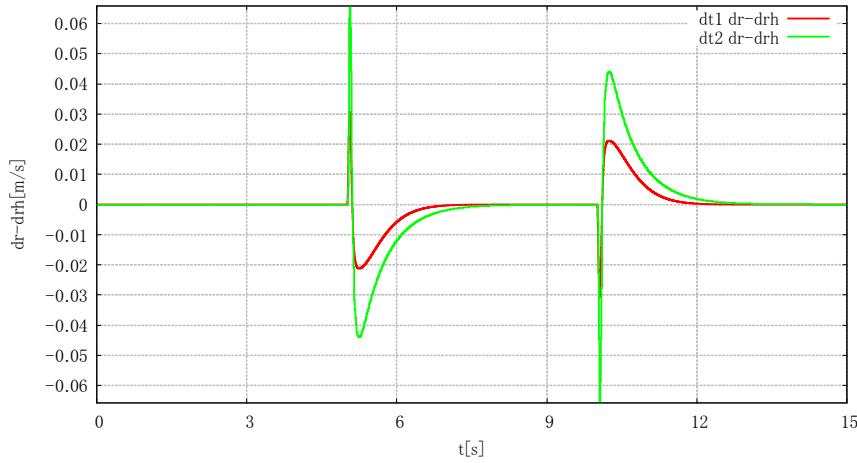


図 4.9: サンプリング周期での推定誤差の比較（台車の速度）

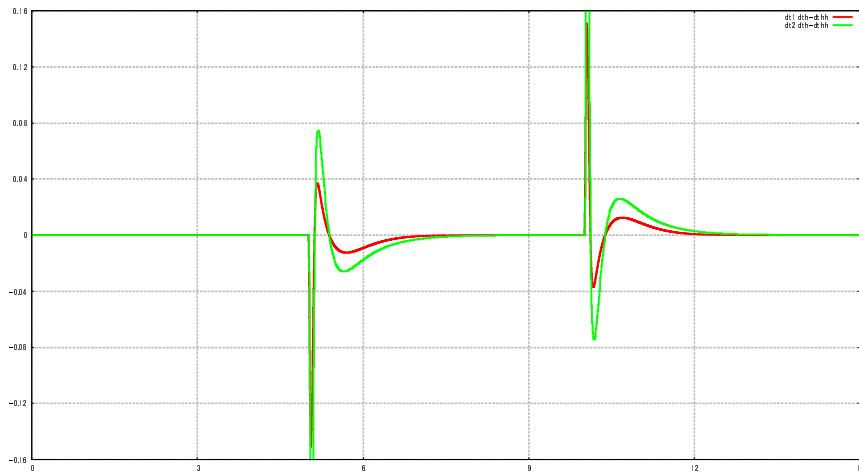


図 4.10: サンプリング周期での推定誤差の比較（振子の各速度）

図4.9と図4.10よりサンプリング周期の短いほうが推定誤差が小さいことが確認できる。以上からサンプリング周期の調整については以下のことが言える。

- サンプリング周期は短いほうが推定誤差が小さくなり、応答がよくなる。
- ただし、あまり小さくしそぎるのはよくない。

4.4 振り上げ制御及び安定化に対する制御性能評価

前節までにおいて、安定化制御における有効なパラメータを考察してきた。ここでは、振り上げ制御におけるパラメータ k, n を調整することで振り上げ制御の制御性能を考察する。

この時、安定化制御に用いるパラメータは以下の表の通りである。安定化制御の際に使用するパラメータについてはこれまでの考察に基づいて決定した。

表 4.4: 振り上げ後の安定化制御の際に用いるパラメータの組

重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
$Q_1:\text{diag}(1\text{E}5, 1\text{E}5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$

振り上げ制御に用いる各パターンのパラメータは以下の表の通りである。

表 4.5: 振り上げ制御に用いるパラメータの組

	n	k
パターン 1	0.4	1.0×10^3
パターン 2	0.4	1.0×10^4
パターン 3	0.4	1.0×10^5

以上の表のパラメータを用いて行ったシミュレーションの結果を示す。

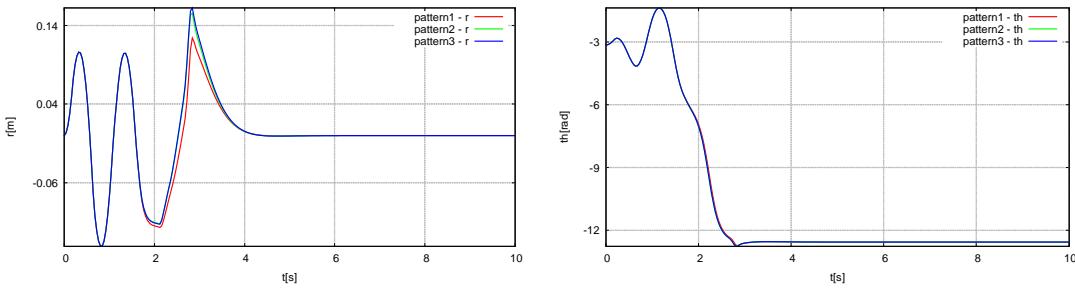


図 4.11: 振り上げ制御のシミュレーション比較結果

図 4.11 よりパラメータ k を変えて振り上げ制御から安定化制御に移行するまでの時間に変化がないことがわかる。前章のシステム解析においては k を大きくすると、エネルギーの収束が早くなると述べた。それにより、振り上げ制御にかかる時間は k を大きくすることで短くなると予想したがそうではなかったようである。左図より、台車の位置に関しては k の値によって若干の差異が確認できるが、有意な違いは出ていない。右図より、振子の角度に関しては k の値による差異はほとんど確認できない。以上から振り上げ制御に関して以下のことが言える。また、

- k の値にかかわらず振り上げ制御から安定化制御に移行するまでの時間は一定である。
- k の値にかかわらず安定化制御までの台車の位置と振子の角度はほぼ同じ挙動を示す。

ちなみに、右図より、振子の角度が -12 より小さい値で収束していることから、このシミュレーションでは、振子は一回転半してから安定化制御に移ったと考えられる。

第5章 実験

5.1 安定化制御実験

振子を上向きに配置したときに安定化制御が可能か実験を行う。(実験目的の第一項目)その際に、振子を真上に配置するのではなく、幾分か角度をつけてから実験を始める。実験に用いたパラメータを以下の表に示す。

表 5.1: 安定化制御実験で使用したパラメータの組

重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$

以下に実験結果とシミュレーション結果の比較した図を示す。

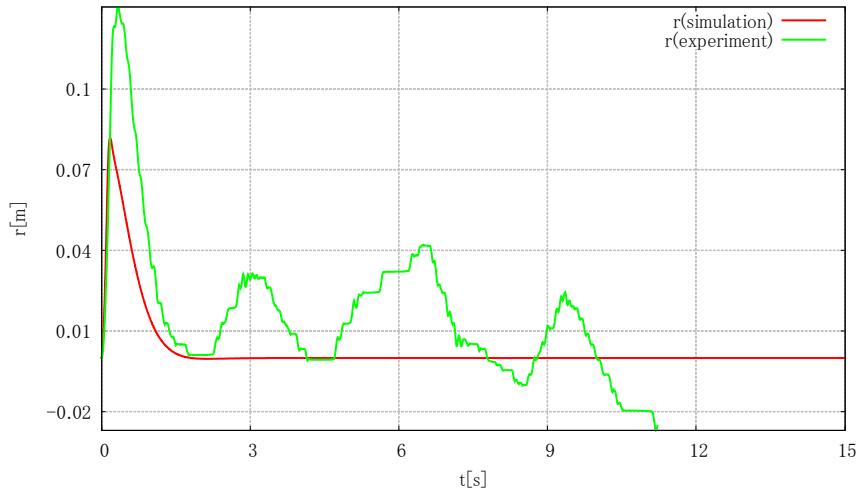


図 5.1: 安定化制御実験結果 (台車位置)

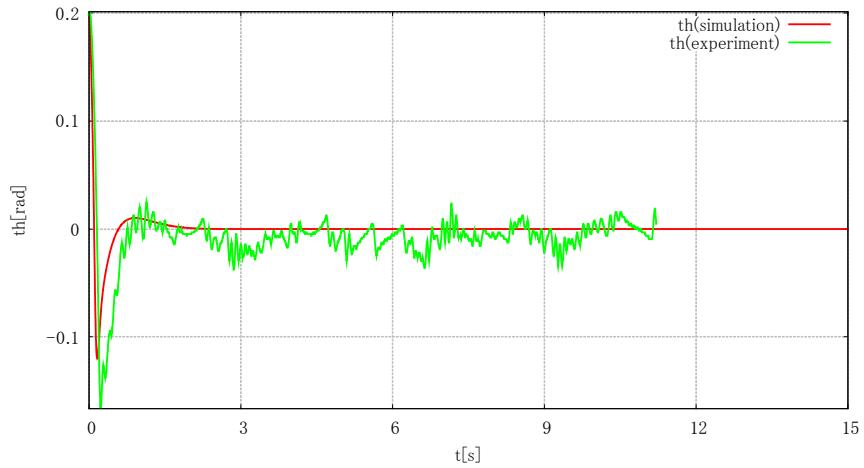


図 5.2: 安定化制御実験結果（振子角度）

図 5.1 と図 5.2 より一番最初の山は実験のほうが大幅に大きくなっているのと、ノイズが乗っている点を除いてシミュレーションと実験結果は概ね一致しているといえる。これは、実験のほうではデータの計測を開始して振子から手を離したためその影響が出たといえる。この時の初期角度は 11.34° である。よって、初期角度がある程度ある中で実験を開始し、安定化制御を行うことができたため、実験目的の第一項目を達成できたといえる。

5.2 目標値の変更実験

台車に目標値を与えて、その目標値に台車が移動しても安定化制御可能か実験を行う。(実験項目の第二項目) なお、目標値は5秒ごとに $0 \rightarrow 0.1 \rightarrow 0$ のように変更される。以下に、重み行列を変更した場合の比較、オブザーバの極を変更した場合の比較、サンプリング周期を変更した場合の比較を行った図を示す。

5.2.1 重み行列の違いによる実験結果の比較

最初に重み行列を変更した場合の実験結果の比較を見てみる。

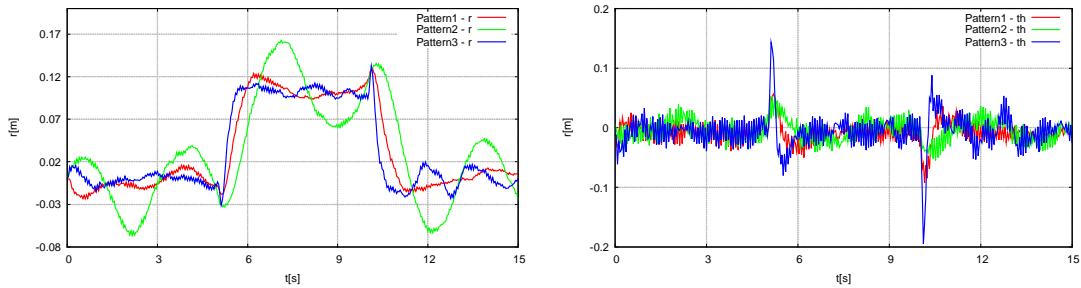


図 5.3: 重み行列の違いによる実験結果の比較

図中の Pattern は以下の表に対応するパラメータである。

表 5.2: パターンとパラメータの対応表

パターン	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
Pattern1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
Pattern2	$Q_1:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
Pattern3	$Q_1:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$

図 5.3 の左図を見ると、台車の位置に関しては、パターン 3 → パターン 1 → パターン 2 の純に応答が遅くなっていることがわかる。特にパターン 2 の方は目標値変更に対して台車が移動してはいるが、波打って動いていることがわかる。つまり、重み行列の第一成分を大きくしているのパターン 3 の応答が一番よく、重み行列の第二成分を大きくしたパターン 2 はバランスがくずれてしまい応答が一番悪いといえる。右図の振子の角度についても同様のことが言える。パターン 3 は台車が動く時に角度が大きく傾いてしまっているが、パターン 2 は台車が動く時においても、大きな角度の傾きは確認できない。つまり、重み行列の第二成分を大きくしているパターン 2 の応答が一番よく、重み行列の第一成分を大きくしたパターン 3 はバランスが崩れてしまい応答が一番悪いといえる。シミュレー

ションの章でも述べたように重み行列の各成分は各状態に対応しており、応答を良くしたい状態があれば、それに対応する成分をおおきくすればよいということが実験でも確認できる。

5.2.2 オブザーバの極の違いによる実験結果の比較

次にオブザーバの極の違いによる実験結果の比較を見てみる。

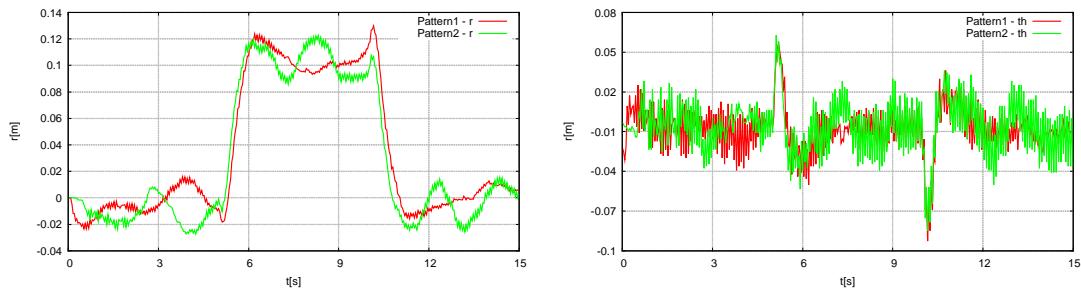


図 5.4: オブザーバの極の違いによる実験結果の比較

図中の Pattern は以下の表に対応するパラメータである。

表 5.3: パターンとパラメータの対応表

パターン	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
Pattern1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
Pattern2	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$

図 5.4 の左図より、台車の位置についてはパターン 1 とパターン 2 では大きな違いは確認できないといえる。右図より、振子の角度についても、パターン 1 とパターン 2 では大きな違いは確認できないといえる。

5.2.3 サンプリング周期の違いによる実験結果の比較

次にサンプリング周期の違いによる実験結果の比較を見てみる。

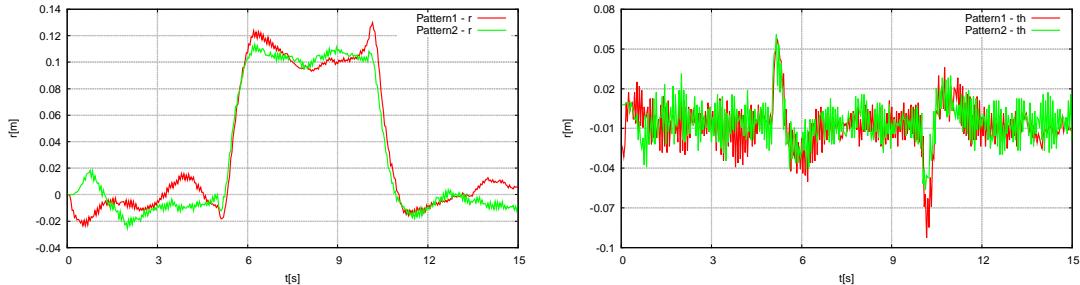


図 5.5: サンプリング周期の違いによる実験結果の比較

図中の Pattern は以下の表に対応するパラメータである。

表 5.4: パターンとパラメータの対応表

パターン	重み行列 Q	オブザーバの極 P	サンプリング周期 Δ [s]
Pattern1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
Pattern2	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.010$

図 5.5 の左図より、台車の位置については 2 パターンとも大きな違いはないといえる。右図より、振子の角度についても 2 パターンとも大きな違いはないといえる。このことから概ねシミュレーションと同じ結果を得ることができたといえる。

5.2.4 シミュレーション結果と実験結果の比較

以下に実験結果とシミュレーション結果との比較を行った図を示す。また、図の数が多いので図のキャプションとその図における各パラメータの対応表を示す。

表 5.5: 図に対応するパラメータの組

図のキャプション	重み行列 Q	オブザーバの極 P	サンプリング周期 $\Delta [s]$
比較結果その 1	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
比較結果その 2	$Q_1:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
比較結果その 3	$Q_1:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.005$
比較結果その 4	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
比較結果その 5	$Q_1:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
比較結果その 6	$Q_1:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.005$
比較結果その 7	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.010$
比較結果その 8	$Q_1:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.010$
比較結果その 9	$Q_1:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-30, 0), (-30, 0))'$	$\Delta_1:0.010$
比較結果その 10	$Q_1:\text{diag}(1E5, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.010$
比較結果その 11	$Q_1:\text{diag}(1E5, 1E6, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.010$
比較結果その 12	$Q_1:\text{diag}(1E6, 1E5, 1, 1)$	$P_1:((-60, 0), (-60, 0))'$	$\Delta_1:0.010$

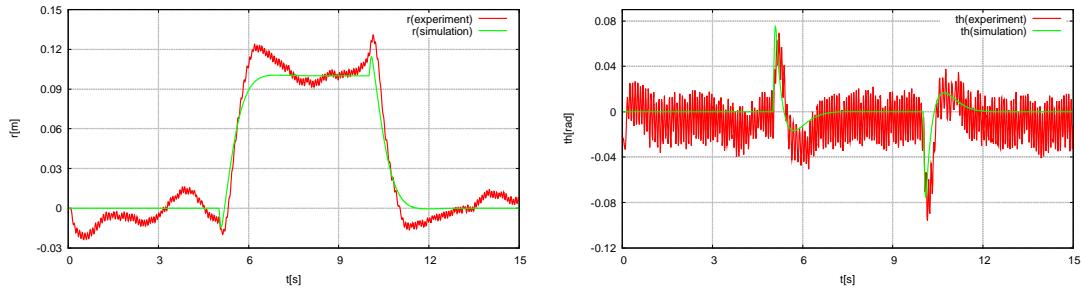
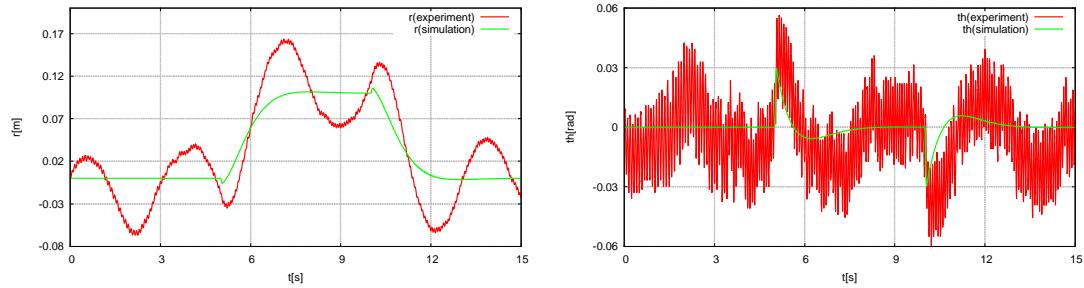
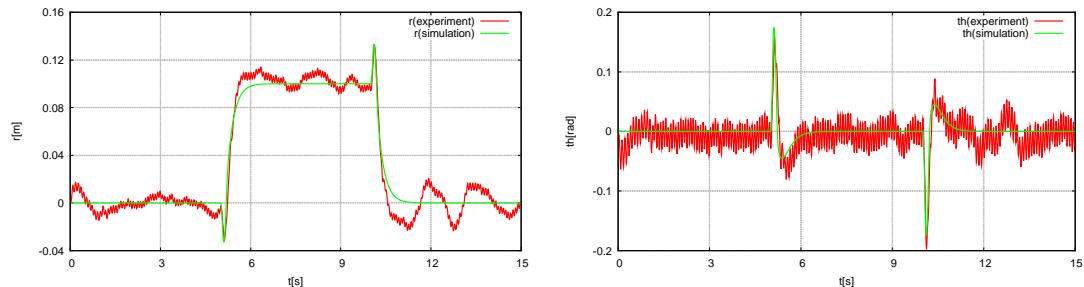
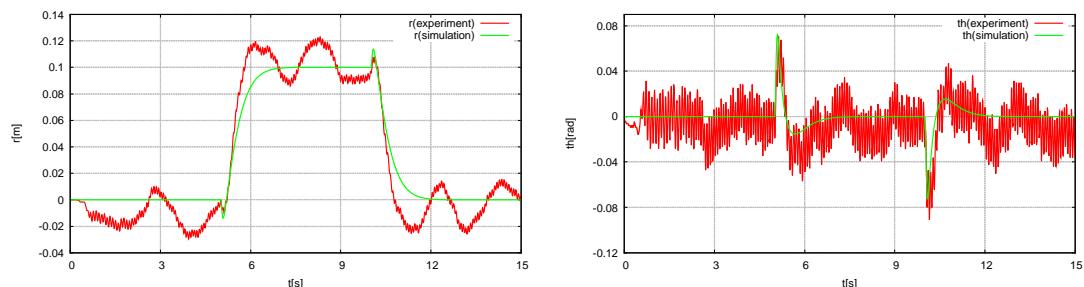
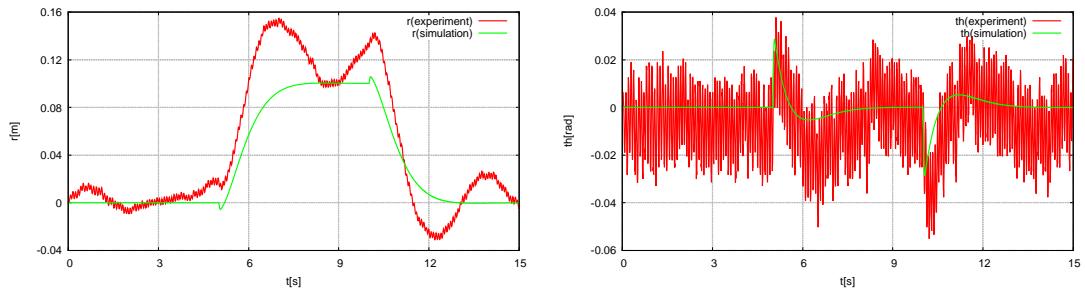
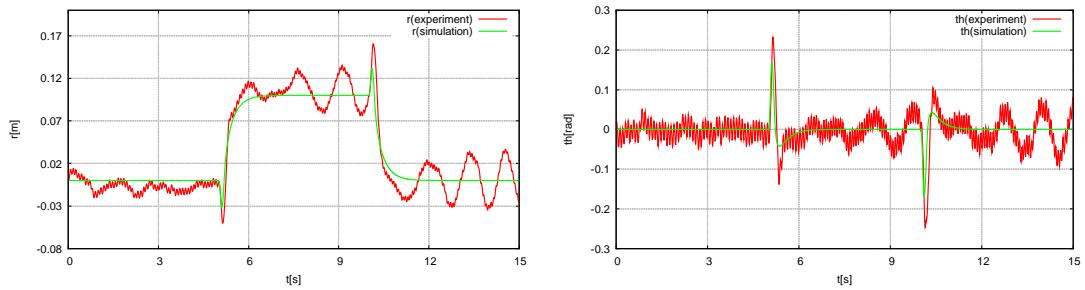
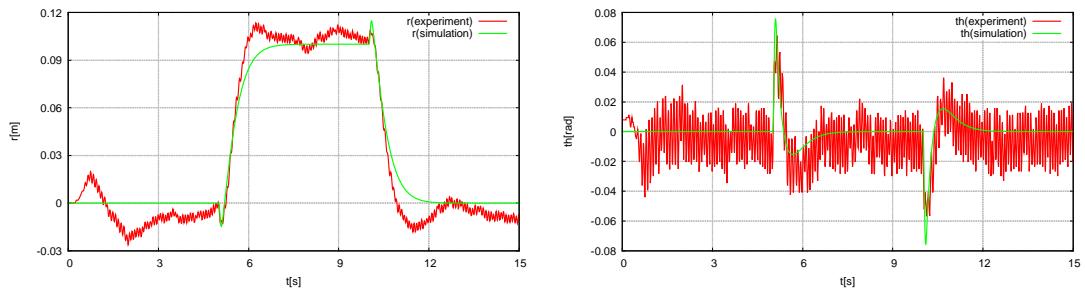
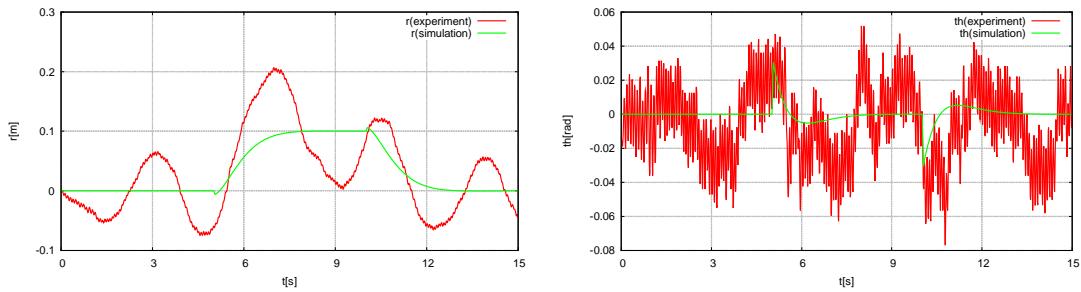
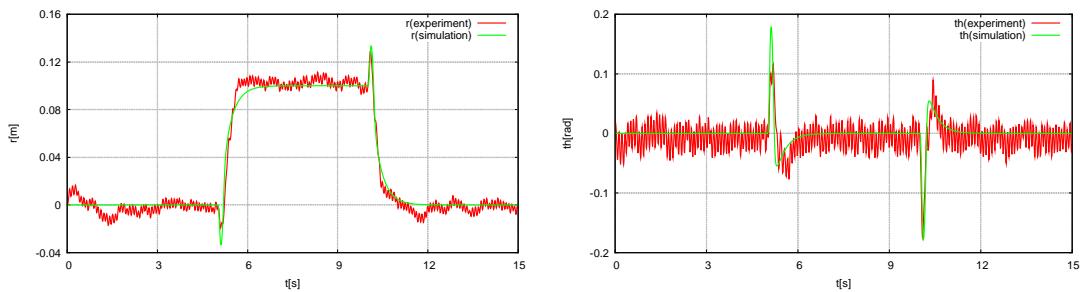
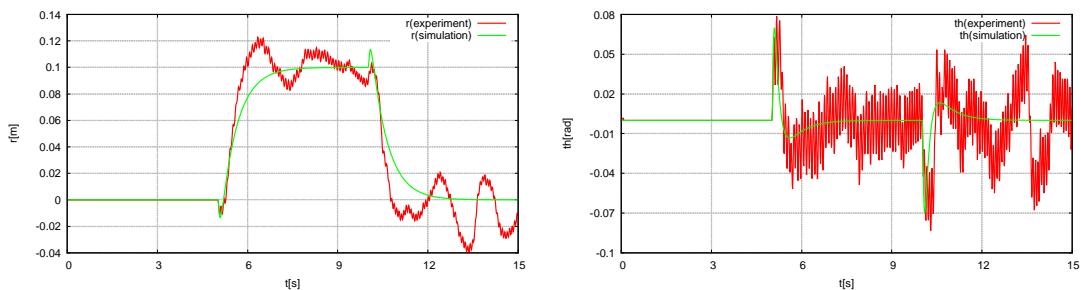
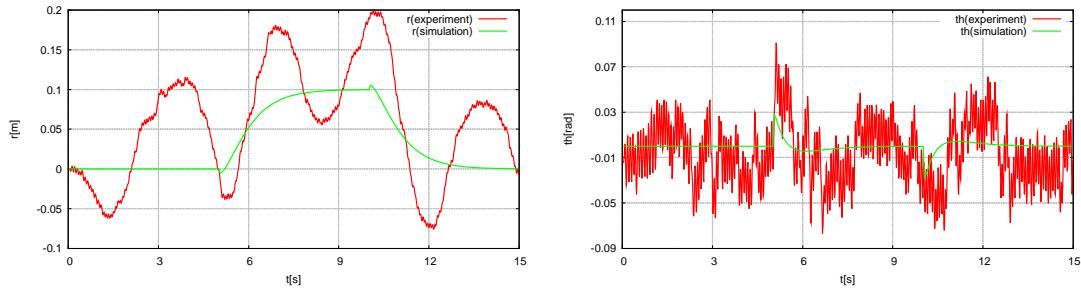
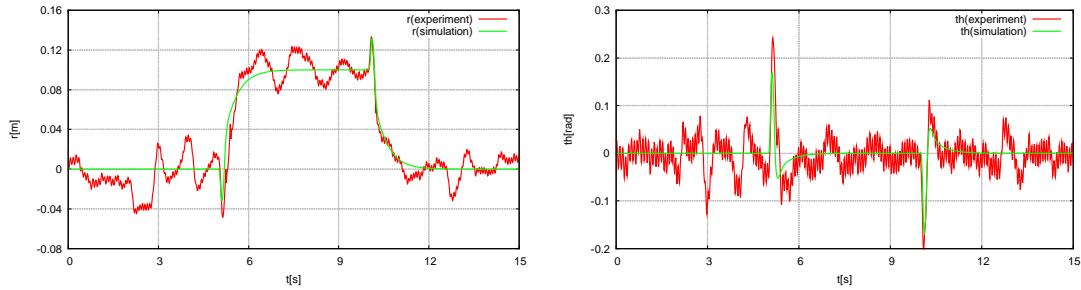


図 5.6: 比較結果その 1 (左図が r , 右図が θ)

図 5.7: 比較結果その 2(左図が r , 右図が θ)図 5.8: 比較結果その 3(左図が r , 右図が θ)図 5.9: 比較結果その 4(左図が r , 右図が θ)

図 5.10: 比較結果その 5(左図が r , 右図が θ)図 5.11: 比較結果その 6(左図が r , 右図が θ)図 5.12: 比較結果その 7(左図が r , 右図が θ)

図 5.13: 比較結果その 8(左図が r , 右図が θ)図 5.14: 比較結果その 9(左図が r , 右図が θ)図 5.15: 比較結果その 10(左図が r , 右図が θ)

図 5.16: 比較結果その 11(左図が r , 右図が θ)図 5.17: 比較結果その 12(左図が r , 右図が θ)

以上の図を見てみるとすべての図においてノイズがひどいが、比較的シミュレーション結果と一致している図とまったく一致していない図があるのが分かる。前者に当たる図は図 5.6、図 5.8、図 5.9、図 5.11、図 5.12、図 5.14、図 5.17。後者に当たる図は、図 5.7、図 5.10、図 5.13、図 5.16 である。後者に共通する特徴は重み行列が $\text{diag}(1E5, 1E6, 1, 1)$ となっている点である。このようにしたときの特徴としてはシミュレーションの章でも述べたが、大きくした成分に対応する状態の応答がよくなるというものであった。つまり、この重み行列のとき θ の応答がよくなりはるはずである。だが、実際の図を見てみると θ の応答はシミュレーションと比較しても一致しているとは言えない。また、 r に関しては図を見る限りでは目標値変更ができていないといえる。これは、重み行列をこうしたことで r の応答は遅くなつたためといえる。普通であれば、プログラムが 5 秒おきに目標値を変更するので、台車はその目標値に向かって動き出す。しかし、 r の応答が遅いため台車は目標値に到達することができず、プログラムから次の目標値が入力される。こうなってしまうことで台車は常に動き続けている状態になり、応答が良くなるはずの θ もそれに伴い悪くなるといえる。そのため、 r は目標値に追従できず、 θ の応答も悪くなるといえる。

シミュレーションと実験において差異がでたが目標値変更を行つての安定化制御を行うことができたので実験目的の第二項目は達成できたといえる。

5.3 振り上げ制御及び安定化実験

振子を真下に配置し、そこから台車の動きだけで振子を振り上げ、安定化制御が可能か実験を行う（実験項目の第三項目）。以下に k を変更したときの実験結果を示す。

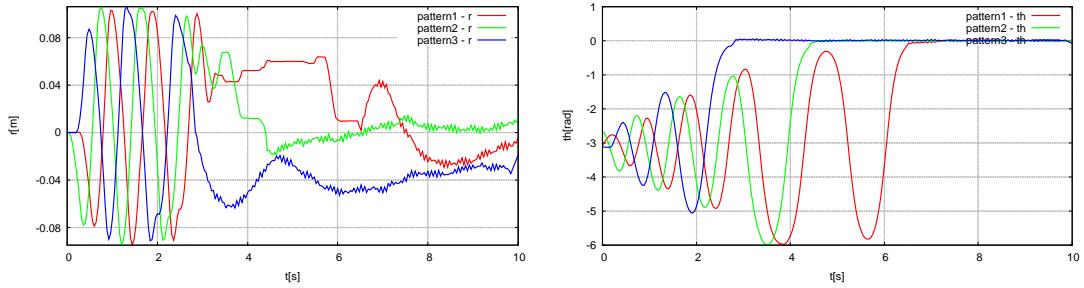


図 5.18: k の違いによる実験結果の比較

図中の Pattern は以下の表に対応するパラメータである。

表 5.6: 実験に用いたパラメータの組

	n	k
パターン 1	0.4	1.0×10^3
パターン 2	0.4	1.0×10^4
パターン 3	0.4	1.0×10^5

図 5.18 の右図より、 k の値が大きいほど早く安定化制御に移行していることがわかる。 k が大きくなるとエネルギーの収束が早くなるため、より早く安定化制御に移行できたといえる。つまり、シミュレーションの章で行った考察は間違っていたということになる。これは、実験に用いた倒立振子系のシミュレーションを行う際に考慮すべき項目が抜けていたか、同定した倒立振子系のパラメータが間違っていたかなど様々な原因が考えられる。以下に実験結果とシミュレーション結果との比較を行った図を示す。

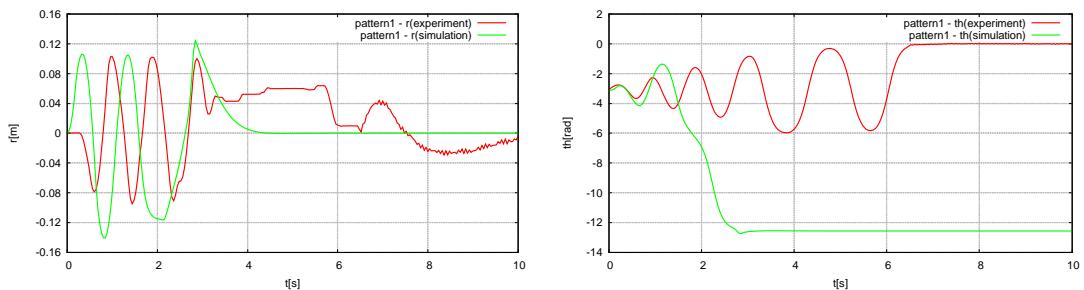


図 5.19: 比較結果 (Pattern1)

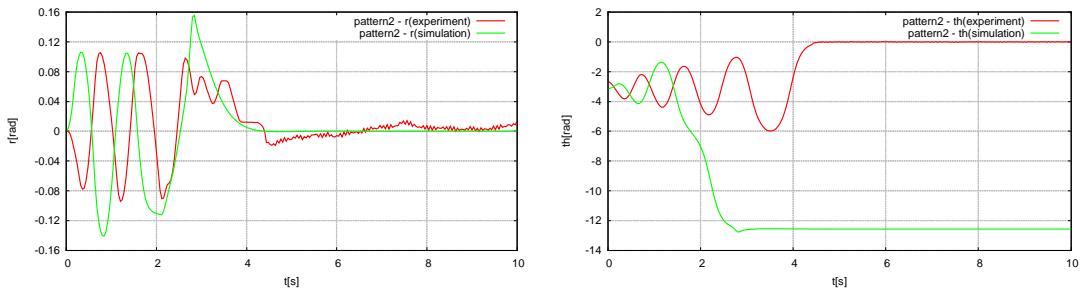


図 5.20: 比較結果 (Pattern2)

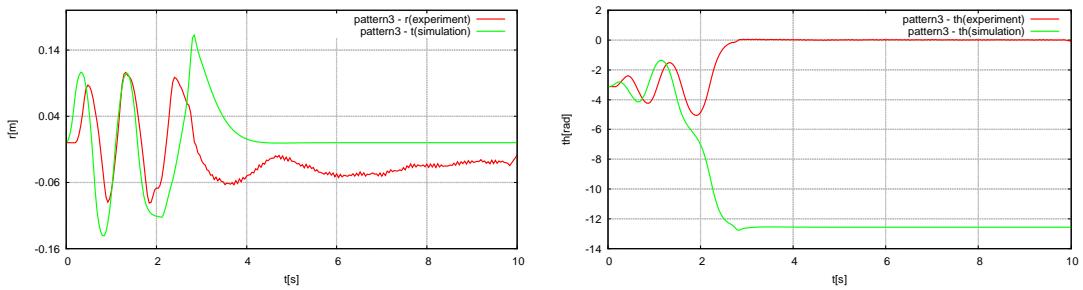


図 5.21: 比較結果 (Pattern3)

シミュレーションの結果と実験の結果に大きな違いが出てしまったが、実験において振り上げ制御から安定化制御を行うことができたので、実験目的の第三項目を達成できたといえる。

第6章 おわりに

本実験を通して当初の目的である、「倒立振子の安定化制御の制御系の設計を状態空間法を用いて行うことにより、線形時不变システムを設計すること」を達成できたといえる。しかし、振り上げ実験においてはシミュレーションの結果と実験の結果に大きな差異が出てきてしまった。そのため、実験を行う前と実験を行った後ではシミュレーションに対する評価が大きく変化した。今後シミュレーションを使用する機会があれば、その結果通りに実験結果が出てくることはないことを考慮した上で使用していきたい。また、制御系のツールや数値計算ツールなどの使い方も習得することができた。

第7章 感想

倒立振子実験は学部3年生の実験の講義で選択しなかったので、今回初めてでした。倒立振子の安定化制御のための理論をプログラムで表現し、それが予想していた結果を射同じ結果を見ることが出来て面白かったです。また、今回はjavaだけでなくMATX、JAMOXの3通りからのアプローチで安定化制御の実現を考えました。それぞれ労力ややりやすさに違いがありプログラミング言語には適材適所があることを改めて知ることができました。

今回の実験で一番勉強になったのは、シミュレーションと実験は必ずしも一致しないということです。今まで、シミュレーションがうまくいけば実験でもうまくいくものだと決めつけていました。ですが、実際はそうではなく、うまくいく可能性はあるけどうまくいかない可能性もあるということでした。また、たとえシミュレーションでうまくいっても実験でやってみるとまくいくが同じような結果にならないということも経験しました。シミュレーションはあくまでコンピュータの中で計算した結果であり、現実世界の事象として発現したわけではないことを十分に理解したうえでシミュレーションを使っていきたいと思います。

関連図書

- [1] 古賀雅伸. 制御工学実験第3 倒立振子の安定化制御.
- [2] 古賀雅伸. B. 倒立振子の安定化制御.

付録A プログラム

A.1 非線形モデル

倒立振子の非線形モデルのコードを載せる。下向きを基準としている。

```
diff_eqs -
```

```

    Func Matrix diff_eqs(t,x,u)
    Real t;
    Matrix x,u;
    {
        Real r,th,dr,dth;
        Matrix xp,dxp;
        Matrix K,KZ;

        r=x(1,1);
        th=x(2,1);
        dr=x(3,1);
        dth=x(4,1);

        K=[[M + m , m *l*cos(th)]
            [m*l*cos(th) , J + m*l*l]];
        KZ=[[-f*dr+ m*l*sin(th)*dth*dth + a * u(1,1)]
            [ m*g*l*sin(th) - c*dth ]];
        dxp=[[x(3:4,1)] [K\KZ]]; // 倒立振子の状態の微分（非線形モデル）

        cnt = cnt+1;
        XX(1,cnt) = dxp(1,1)-Xh(1,1);

        // 出力を更新
        return dxp;//Xは6行の行列になる
    }
}
```

A.2 線形モデル

倒立振子の線形モデルのコードを載せる。下向きを基準としている。

```
main. -
```

```

    Func void diff_eqs(DX,t,X,UY)
    Real t;
    Matrix X,DX,UY;
    {
```

```

Real M,m,l,J,f,a,c,g;
Matrix xp,up,dxp;
Matrix A,B,A21,A22,B2,K;

//物理パラメータの設定
M=1.49; m=0.038; l=0.13;
J=4.5E-4; f=15.10; a=0.73;
c=2.1E-4; g=9.8;

K=[[M+m,m*l][m*l,J+m*l^2]];
A21 = K~*[[0,0][0,m*g*l]];
A22 = K~*[[ -f,0][0,-c]];
A=[[Z(2),I(2)][A21,A22]];

B2 = K~*[[a][0]];
B=[[Z(2,1)][B2]];

xp = X;
up = UY;

dxp = A*xp+B*up;
DX = [dxp];
}

```

A.3 システム解析

第4章4.1節のシステム解析に用いたコードを載せる。ただし、行列A,B,Cについては別の部分で計算している。

```

main.c
    //システムの極(Aの固有値)を計算
    D=eigval(A);

    //可制御性行列のランクを計算し、可制御性を調べる
    Nc=[B,A*B,A*A*B,A*A*A*B];

    //可観測性行列もランクを計算し、可観測性を調べる。
    No=[[C][C*A][C*A*A][C*A*A*A]];

```

A.4 状態フィードバック

状態フィードバックを設計算するのに用いたコードを載せる。ただし、LQ最適制御に基づくフィードバック則とする。

```

lqr
    Matrix A, B, Q, R, F, P;

```

```

Q = diag(1, 1, 1, 1);
R = [1];
{F,P} = lqr(A, B, Q, R);

```

A.5 最小次元オブザーバ

最小次元オブザーバをゴピナスの方法で計算するのに用いたコードを載せる。

```

obsg —————
CoMatrix obs_p;
obs_p = trans([(-2,0), (-2,0)]);
{Ah, Bh, Ch, Dh, Jh} = obsg(A, B, C, obs_p);

```

A.6 コントローラの離散化

連続時間オブザーバを離散化するたために用いたコードを載せる。

```

c2d —————
// ここは関数の先頭
Real dt;
Matrix Ah,Bh,Jh;
Matrix Ahd,Bhd,Jhd,Hhd;
// ここはオブザーバの設計の後
dt = 0.005;
{Ahd,Hhd} = c2d(Ah, [Bh Jh], dt);
Bhd = Hhd(:,1:2);
Jhd = Hhd(:,3);

```

A.7 シミュレーション

第5章でシミュレーションを行ったコードを載せる。

```

InPeAboveNonLinerDesign3.mm —————
Matrix A; //システム行列
Matrix B; //入力行列
Matrix C; //出力行列
Matrix F; //状態フィードバック行列
Matrix z; // オブザーバの状態
Matrix Ah,Bh,Ch,Dh,Jh; //オブザーバに関する行列
Matrix Ahd,Bhd,Jhd;
Matrix Xh,XX,Xtest; //推定値

```

```

Real M,m,l,J,f,a,c,g,c1,c2; //パラメータ

Func void main()
{
    Real t0,t1,r0,th0,tol;
    Real dt,dtsav; //離散化に用いる変数
    Matrix x0,z0,xp0,TC,XC,UC;
    void calcParameter();
    Matrix diff_eqs(),link_eqs();

    t0 = 0.0; //シミュレーション開始時刻
    t1 = 18.0; //シミュレーション終了時刻
    r0 = 0; //位置の初期値
    th0 = 10.0; //角度の初期値
    x0 = [r0, th0/180*PI ,0 ,0]'; //倒立振子の初期状態,
    z = [0,0]'; //オブザーバーの初期状態,
    dt = 0.005; //サンプリング周期
    tol=1.0E-9;//許容誤差
    dtsav = 0.05; //データ保存間隔

    print "Now simulating\n";

    cnt=0; // 初期化
    calcParameter(); //シミュレーションに必要なパラメータを計算
    // Ode() によってシミュレーションを行う
    {TC,XC} =
        Ode45HybridAuto(t0,t1,dt,x0,diff_eqs,link_eqs,tol,dtsav);
    // diff_eqs() は微分方程式を記述する関数
    // link_eqs() は複数の微分方程式の関係を記述する関数。
    // TC:時間の時系列
    // XC:状態 x(t) の時系列
    // UC:入出力ベクトルの時系列

    //数値回を表示する。横軸 t, 縦軸 x(t)
    mgrepplot(1,TC,XC(1,*),{"r"});
    mgrepplot(2,TC,XC(2,*),{"Theta"});

    print TC >> "TC.mat";
    print XC >> "XC.mat";
    print UC >> "UC.mat";
}

// diff_eqs() は微分方程式を記述する関数
Func Matrix diff_eqs(t,x,u)
// t は時間
Real t;
Matrix x,u;
{
    Real r,th,dr,dth;
    Matrix xp,dxp;
    Matrix K,KZ;

    r=x(1,1);
    th=x(2,1);
    dr=x(3,1);
}

```

```

dth=x(4,1);

K=[[M + m , m *l*cos(th)]
   [m*l*cos(th) , J + m*l*l]];
KZ=[[-f*dr+ m*l*sin(th)*dth*dth + a * u(1,1)]
   [ m*g*l*sin(th) - c*dth ]];
dxp=[[x(3:4,1)] [K\KZ]]; // 倒立振子の状態の微分（非線形モデル）

cnt = cnt+1;
XX(1,cnt) = dxp(1,1)-Xh(1,1);

// 出力を更新
return dxp;//Xは6行の行列になる
}

// link_eqs() は複数の微分方程式の関係を記述する関数。
Func Matrix link_eqs(t,x)
Real t;
Matrix x;
{
    Matrix u;
    Matrix xref;
    Matrix xh,y;

    //台車の可動範囲に関する制限
    if (x(1,1) <= -0.16 || 0.16 <= x(1,1)) { // r=x(1,1)
        OdeStop();
    }

    y = C*x; // 出力の計算
    xh = Ch*z + Dh*y; // 状態の推定値
    Xh= xh; //推定値を保存
    xref = [0,0,0,0]'; // 状態の目標値,
    if(0<=t && t<=5){
        xref = [0,0,0,0]'; // 状態の目標値,
    }else if(5<t&&t<=10){
        xref = [0.1,0,0,0]'; // ,
    }else if(10<t&&t<=15){
        xref = [0,0,0,0]'; // ,
    }
    u = F*(xref - xh);

    //入力の大きさに関する制限
    if(u(1,1) <= -15) {
        u(1,1) = -15;
    }else if (u(1,1) >= 15) {
        u(1,1) = 15;
    }

    z = Ahd*z + Bhd*y + Jhd*u; // オブザーバの状態更新
    //入力を更新
    return u;
}

```

```

Func void calcParameter(){

    CoMatrix pc,obs_p;
    Matrix A21,A22,B2,K,N;
    Matrix Ah ,Bh ,Jh ; // 連続時間オブザーバの係数行列
    Matrix Hhd; // 離散時間オブザーバの係数行列
    Real dt; //サンプリング周期
    Matrix Q,R,P;

    //物理パラメータの設定
    M=0.69; m=0.031; l=0.15;
    J=2.5E-4; f=7.6; a=0.61;
    c=5.4E-5; g=9.8; c1=1.0;
    c2=1.0;

    // システム行列 A の準備
    K=[[M+m,m*1][m*1,J+m*1^2]];
    A21 = K~*[[0,0][0,m*g*1]];
    A22 = K~*[[ -f,0][0,-c]];
    A=[[Z(2),I(2)][A21,A22]];

    //入力行列 B の準備
    B2 = K~*[[a][0]];
    B=[[Z(2,1)][B2]];

    //出力行列 C の準備
    N=[[c1,0][0,c2]];
    C=[N,Z(2)];

    //状態フィードバック行列 F を準備する
    //pc=[(-100,0),(-50,0),(-1,0),(-1,0)]'; 閉ループ系の極,
    //F = place(A,B,pc);

    //LQ最適制御のための状態フィードバック行列 F の準備
    Q = diag(1E5,1E5,1,1);
    R = [1];
    {F,P} = lqr(A, B, Q, R);

    //ゴピナスの方法で最初次元観測器を設計する
    obs_p = trans([(-30,0), (-30,0)]); // オブザーバの極
    //read obs_p; // 極を編集する
    {Ah, Bh, Ch, Dh, Jh} = obsg(A, B, C, obs_p); // ゴピナスの方法による
    設計

    // 連続時間オブザーバをサンプリング周期で離散化
    dt = 0.005;
    {Ahd,Hhd} = c2d(Ah,[Bh, Jh], dt); //離散化
    Bhd = Hhd(:,1:2); //係数行列の取り出し
    Jhd = Hhd(:,3); // 係数行列の取り出し
}

```

A.8 安定化制御及び目標値変更実験

安定化制御及び目標値変更実験を行うために使用したコードを載せる。

sample.mm

```
#define LOGMAX 10000

Integer cmd, count;
Real smtime;
Matrix u, y;
Array data;
Matrix Ahd,Bhd,Chd,Dhd,Jhd,F;
Matrix z;
Integer qrr,qth,oprh,opthh;
Real ref;

// センサとアクチュエータ関連の変数 (hardware.mm で使用される)
Matrix mp_data, PtoMR;

// メイン関数
Func void main()
{
    void para_init(), var_init();
    void on_task(), break_task(), off_task_loop();
    void machine_ready(), machine_stop(), data_save();
    void calcParameter();

    para_init();           // パラメータの初期化
    var_init();            // 変数の初期化
    machine_ready();       // 実験装置の準備
    calcParameter();

    rtSetClock(smtime);   // サンプリング周期の設定
    rtSetTask(on_task);   // オンライン関数の設定(制御)
    rtSetBreak(break_task); // 割り込みキーに対応する関数の設定

    rtStart();             // リアルタイム制御開始
    off_task_loop();       // オフライン関数
    rtStop();              // リアルタイム制御終了

    machine_stop();        // 実験装置を停止
    data_save();            // データを保存する
}

// パラメータの初期化
Func void para_init()
{
}

// 変数の初期化
Func void var_init()
{
    smtime = 0.01;         // サンプリング周期 [s]
    cmd = 0;                // 制御出力を抑制
    count = 0;               // ロギングデータの数
    data = Z(4,LOGMAX); // ロギングデータを保存する場所
```

```

z = [0 0]'; // オブザーバの初期値
qrr = 0;
qth = 0;
oprh = 0;
opthh = 0;
ref = -0;
}

// オンライン関数
Func void on_task()
{
    Matrix xh,xref;
    Matrix sensor();
    void actuator();

    y = sensor(); // センサから入力

    xh = Chd*z + Dhd*y; // 状態の推定値
    if(count * smtime < 5){
        ref = 0;
    } else if(count * smtime < 10){
        ref = 0.1;
    } else if (count * smtime < 15){
        ref = 0;
    }

    xref = [ref 0 0 0]'; // 状態の目標値

    u = F*(xref - xh); // 制御入力
    z = Ahd*z + Bhd*y + Jhd*u; // オブザーバの状態更新

    // リハーサル中でなければ
    if (cmd == 1 && ! rtIsRehearsal()) {
        actuator(u(1)); // アクチュエータへ出力
    }

    // データのロギング
    if (cmd == 1 && count < LOGMAX) {
        count++;
        data(1:1, count) = u; // 入力
        data(2:3, count) = y; // 実測値（台車一、振り子角度）
        data(4, count) = ref;
    }
}

// オフライン関数
Func void off_task_loop()
{
    Integer end_flag;

    end_flag = 0;

    gotoxy(5, 6);
    printf("'c': アクチュエータへ出力");
    gotoxy(5, 7);
    printf("ESC: アクチュエータへ出力停止");
    gotoxy(5, 8);
    printf("r:目標値の変更! ");
}

```

```

do {
    gotoxy(5, 11);
    printf("  台車位置 = %8.4f [m], 振子角度 = %8.4f [deg]",
           y(1), y(2)/PI*180);
    gotoxy(5, 12);
    printf("入力 = %10.4f [N]", u(1));

    gotoxy(5, 14);
    printf("データ数 = %4d,
           時間 = %7.3f [s] ref = %4f", count, count*smtime,ref);
    if (rtIsTimeOut()) {
        gotoxy(5, 18);
        warning("\n時間切れ !\n");
        break;
    }

    if (kbhit()) {
        switch (getch()) {
            case 0x1b:          /* ESC */
                end_flag = 1;
                break;
/* 'c' */ case 0x43: // アクチュエータへ出力開始
/* 'C' */ case 0x63: // If 'c' or 'C' is
                    // pressed, start motor
                cmd = 1;
                break;
            case 0x72:
                gotoxy(5,16);
                print "台車の目標値「m」 :"; read ref;
                gotoxy(5,16);
                printf("          ");
                break;
            default:
                break;
        }
    }
} while ( ! end_flag); // If end_flag != 0, END
}

// 割り込みキーに対応する関数
Func void break_task()
{
    void machine_stop();

    rtStop();
    machine_stop(); // 実験装置停止
}

// 実験装置の準備
Func void machine_ready()
{
    void sensor_init(), actuator_init();

    sensor_init();           // センサの初期化
    actuator_init();         // アクチュエータの初期化
}

```

```

gotoxy(5,5);
printf("台車の初期位置 : レールの中央");
gotoxy(5,6);
printf("振子の初期位置 : 真下");
gotoxy(5,9);
pause "台車と振子を初期位置に移動し, リターンキーを入力して下さい。";
gotoxy(5,9);
printf("                                         ");
gotoxy(5,9);
pause "リターンキーを入力すると, 制御が開始されます。";
clear;
}

// 実験装置停止
Func void machine_stop()
{
    void actuator_stop();
    actuator_stop();
}

// データのファイルへの保存
Func void data_save()
{
    String filename;
    Array TT;

    filename = "experiment_Q56obs60dt10";
    read filename;

    if (count > 1) {
        TT = [0:count-1]*smtime;
        print [[TT][data(:,1:count)]] >> filename +".mat";
    }
}

Func void calcParameter(){

    CoMatrix pc,obs_p;
    Matrix A21,A22,B2,K,N;
    Matrix Ah ,Bh ,Jh ; // 連續時間オブザーバの係数行列
    Matrix Hhd; // 離散時間オブザーバの係数行列
    Real dt; //サンプリング周期
    Real M,m,l,J,f,a,c,g,c1,c2; //パラメータ
    Matrix A; //システム行列
    Matrix B; //入力行列
    Matrix C; //出力行列
    Matrix Q,R,P;

    // パラメータの調整
    qrr = 100000;
    qth = 1000000;
    oprh = -60;
    opthh = -60;

    //物理パラメータの設定
}

```

```

M=0.69; m=0.031; l=0.15;
J=2.5E-4; f=7.6; a=0.61;
c=5.4E-5; g=9.8; c1=1.0;
c2=1.0;

// システム行列 A の準備
K=[[M+m,m*l][m*l,J+m*l^2]];
A21 = K~*[[0,0][0,m*g*l]];
A22 = K~*[[ -f,0][0,-c]];
A=[[Z(2),I(2)][A21,A22]];

// 入力行列 B の準備
B2 = K~*[[a][0]];
B=[[Z(2,1)][B2]];

// 出力行列 C の準備
N=[[c1,0][0,c2]];
C=[N,Z(2)];

// 状態フィードバック行列 F を準備する (極配置)
/*pc=[(-100,0),(-50,0),(-2,0),(-3,0)]; // 閉ループ系の極,
F = pplace(A,B,pc);*/

// 状態フィードバック行列 F を準備する (LQ 最適制御)
Q = diag(qrr,qth,1,1); // 対角行列
R = [1];
{F,P} = lqr(A,B,Q,R); // P: リカッティ方程式の解

// ゴピナスの方法で最初次元観測器を設計する
obs_p = trans([(oprh,0), (opthh,0)]); // オブザーバの極
// read obs_p; // 極を編集する
{Ah, Bh, Chd, Dhd, Jh} = obsg(A, B, C, obs_p); // ゴピナスの方法による設計

// 連続時間オブザーバをサンプリング周期で離散化
dt = smtime;
{Ah, Hhd} = c2d(Ah,[Bh, Jh], dt); // 縮小化
Bhd = Hhd(:,1:2); // 係数行列の取り出し
Jhd = Hhd(:,3); // 係数行列の取り出し
print "aa" >> "a.txt";
}

```

A.9 振り上げ制御及び安定化実験

振り上げ制御及び安定化実験を行うために使用したコードを載せる。

```

sample.mm
#define LOGMAX 10000

```

```

Integer cmd, count;
Real smtime;
Matrix u, y;
Array data;
Matrix Aod,Bod,Cod,Dod,F;
Matrix z,yw;
Matrix zde,zdo;
Integer qrr,qth,oprh,opthh;
Integer swinging,cnt;
Real M,m,l,J,f,a,c,g,c1,c2; //パラメータ
Real pre_r,pre_th,pre_dr,pre_dth,pre_ddth;

// センサとアクチュエータ関連の変数 (hardware.mmで使用される)
Matrix mp_data, PtoMR;

// メイン関数
Func void main()
{
    void para_init(), var_init();
    void on_task(), break_task(), off_task_loop();
    void machine_ready(), machine_stop(), data_save();
    void calcParameter();

    para_init();           // パラメータの初期化
    var_init();            // 変数の初期化
    machine_ready();       // 実験装置の準備
    calcParameter();

    rtSetClock(smtime);   // サンプリング周期の設定
    rtSetTask(on_task);   // オンライン関数の設定(制御)
    rtSetBreak(break_task); // 割り込みキーに対応する関数の設定

    rtStart();             // リアルタイム制御開始
    off_task_loop();       // オフライン関数
    rtStop();              // リアルタイム制御終了

    machine_stop();        // 実験装置を停止
    data_save();           // データを保存する
}

// パラメータの初期化
Func void para_init()
{
    zde = [0,PI]'; //'
    zdo = [0,0]'; //'
    swinging = 1;

    //物理パラメータの設定
    M=0.69; m=0.031; l=0.15;
    J=2.5E-4; f=7.6; a=0.61;
    c=5.4E-5; g=9.8; c1=1.0;
    c2=1.0;

    pre_r = 0;
    pre_th = PI;
}

```

```

pre_dr = 0;
pre_dth = 0;
pre_ddth = 0;

//ここでuを初期化すればよいのではないか
}

// 変数の初期化
Func void var_init()
{
    Matrix Swinging();

    smtime = 0.005;      // サンプリング周期 [s]
    cmd = 0;             // 制御出力を抑制
    count = 0;            // ロギングデータの数
    data = Z(3,LOGMAX); // ロギングデータを保存する場所
    z = [0 0]';          // オブザーバの初期値,
    qrr = 0;
    qth =0;
    oprh = 0;
    opthh = 0;
    count = 0;
    u = Swinging([[0] [-PI+0.01] [0] [0] [0] [0] [0] [0]]);

}

// オンライン関数
Func void on_task()
{
    Matrix xh,xref;
    Matrix xhe,xho;
    Matrix sensor(),AngleWrapper(),DiscreteEstimator()
                ,DiscreteObserver(),Swinging();
    void actuator();

    y = sensor();           // センサから入力

    //AngleWrapper 入力:y 出力:yw
    yw = AngleWrapper(y);

    // DiscreteEstimator 入力:yw 出力:xhe
    xhe = DiscreteEstimator(yw);

    // DiscreteObserver 入力:[[u] [yw]] 出力:xho
    xho = DiscreteObserver([[u] [yw]]);

    // Swinging 入力:[[xhe] [xho]] 出力:u
    u = Swinging([[xhe] [xho]]);

    // リハーサル中でなければ
    if (cmd == 1 && ! rtIsRehearsal()) {
        actuator(u(1));           // アクチュエータへ出力
    }

    // データのロギング
    if (cmd == 1 && count < LOGMAX) {
        count++;
        data(1:1, count) = u; // 入力
        data(2:3, count) = y; // 実測値（台車一、振り子角度）
    }
}

```

```

        }

// オフライン関数
Func void off_task_loop()
{
    Integer end_flag;
    end_flag = 0;

    gotoxy(5, 6);
    printf("'c': アクチュエータへ出力");
    gotoxy(5, 7);
    printf("ESC: アクチュエータへ出力停止");

    do {
        gotoxy(5, 11);
        printf("  台車位置 = %8.4f [m], 振子角度 = %8.4f [deg]",
               y(1), yw(2)/PI*180);
        gotoxy(5, 12);
        printf("入力 = %10.4f [N]", u(1));

        gotoxy(5, 14);
        printf("データ数 = %4d, 時間 = %7.3f [s]", count, count*smtime);
        if (rtIsTimeOut()) {
            gotoxy(5, 18);
            warning("\n時間切れ !\n");
            break;
        }

        if (kbhit()) {
            switch (getch()) {
                case 0x1b:           /* ESC */
                    end_flag = 1;
                    break;
                /* 'c' */ case 0x43: // アクチュエータへ出力開始
                /* 'C' */ case 0x63: // If 'c' or 'C' is
                    // pressed, start motor
                    cmd = 1;
                    break;
                default:
                    break;
            }
        }
    } while ( ! end_flag); // If end_flag != 0, END
}

// 割り込みキーに対応する関数
Func void break_task()
{
    void machine_stop();

    rtStop();
    machine_stop(); // 実験装置停止
}

// 実験装置の準備

```

```

Func void machine_ready()
{
    void sensor_init(), actuator_init();
    sensor_init(); // センサの初期化
    actuator_init(); // アクチュエータの初期化

    gotoxy(5,5);
    printf("台車の初期位置 : レールの中央");
    gotoxy(5,6);
    printf("振子の初期位置 : 真下");
    gotoxy(5,9);
    pause "台車と振子を初期位置に移動し, リターンキーを入力して下さい。";
    gotoxy(5,9);
    printf("");
    gotoxy(5,9);
    pause "リターンキーを入力すると, 制御が開始されます。";
    clear;
}

// 実験装置停止
Func void machine_stop()
{
    void actuator_stop();
    actuator_stop();
}

// データのファイルへの保存
Func void data_save()
{
    String filename;
    Array TT;

    filename = "huriage";
    read filename;

    if (count > 1) {
        TT = [0:count-1]*smtime;
        print [[TT][data(:,1:count)]] >> filename +".mat";
    }
}

Func void calcParameter(){

    CoMatrix pc,obs_p;
    Matrix A21,A22,B2,K,N;
    Matrix Ah ,Bh ,Jh ; // 連続時間オブザーバの係数行列
    Matrix Hhd; // 離散時間オブザーバの係数行列
    Real dt; //サンプリング周期
    Matrix A; //システム行列
    Matrix B; //入力行列
    Matrix C; //出力行列
    Matrix Q,R,P;
    Matrix Chd,Dhd;
}

```

```

// パラメータの調整
qrr = 100000;
qth = 100000;
oprh = -30;
opthh = -30;

// システム行列 A の準備
K=[[M+m,m*1][m*1,J+m*1^2]];
A21 = K~*[[0,0][0,m*g*1]];
A22 = K~*[[ -f,0][0,-c]];
A=[[Z(2),I(2)][A21,A22]];

// 入力行列 B の準備
B2 = K~*[[a][0]];
B=[[Z(2,1)][B2]];

// 出力行列 C の準備
N=[[c1,0][0,c2]];
C=[N,Z(2)];

// 状態フィードバック行列 F を準備する (極配置)
/*pc=[(-100,0),(-50,0),(-2,0),(-3,0)]'; //閉ループ系の極,
F = pplace(A,B,pc);*/

// 状態フィードバック行列 F を準備する (LQ 最適制御)
Q = diag(qrr,qth,1,1); // 対角行列
R = [1];
{F,P} = lqr(A,B,Q,R); // P: リカッティ方程式の解

// ゴピナスの方法で最初次元観測器を設計する
obs_p = trans([(oprh,0), (opthh,0)]); // オブザーバの極
// read obs_p; // 極を編集する
{Ah, Bh, Chd, Dhd, Jh} = obsg(A, B, C, obs_p); // ゴピナスの方法による設計

// 連続時間オブザーバをサンプリング周期で離散化
dt = 0.005;
{Aod,Bod} = c2d(Ah, [Jh Bh], dt); // 離散化
Cod = Chd; // 係数行列の取り出し
Dod = [Z(Rows(Dhd), Cols(Jh)), Dhd]; // 係数行列の取り出し
}

Func Matrix AngleWrapper(y)
Matrix y;
{
    Real r,th;
    Matrix yw;

    // 入力: y 出力: y w
    r = y(1,1);
    th = y(2,1);
    while (th < -PI) {
        th = th + 2*PI;
    }

    while (th > PI) {
        th = th - 2*PI;
    }
}

```

```

}

yw = [[r][th]];
return yw;
}

Func Matrix DiscreteEstimator(yw)
Matrix yw;
{
    Matrix xhe;
    xhe = [[yw][(yw-zde)/smtime]];
    zde = yw; //更新
    return xhe;
}

Func Matrix DiscreteObserver(uyw)
Matrix uyw;
{
    Matrix xho;
    Matrix r_th;

    r_th = uyw(2:3,1);
    if(swinging == 1) {
        xho = [[r_th][Z(2,1)]];
    } else {
        xho = Cod*zdo + Dod*uyw;
    }

    if(swinging == 1) { // 更新
        zdo = [0,0]'; //
    } else {
        zdo = Aod*zdo + Bod*uyw;
    }

    return xho;
}

Func Matrix Swinging(xheo)
Matrix xheo;
{
    Real r,th,dr,dth,ddr,ddth;
    Real E0,r_max,th_min;
    Real E;
    Real n,kk;
    Integer isSolverTrial();

    n = 0.4;
    kk = 100000;

    E0 = 0;
    r_max = 0.07;
    th_min = 12.0;
}

```

```

r = xheo(1,1);
th = xheo(2,1);
dr = xheo(3,1);
dth = xheo(4,1);

if(isSolverTrial() == 0
    && abs(th) <= th_min*PI/180 && abs(r) < r_max) {
    swinging = 0;
}
if(isSolverTrial() == 0 && abs(th) > th_min*PI/180){
    swinging = 1;
}

if(swinging == 1){ // 振り上げ制御
    E = (J + m*l*l)*dth*dth/2 + m*g*l*(cos(th) - 1);
    ddr = max(-n*g,min(n*g,kk*(E-E0)*sgn(dth*cos(th))));

    if(isSolverTrial() == 0){
        pre_ddth = ddth = (dth - pre_dth)/smtime;
        pre_dth = dth;
    } else {
        ddth = pre_ddth;
    }

    u = [((M + m)*ddr + m*l*(ddth*cos(th)
        - dth*dth*sin(th)) + f*dr)/a];

    if(abs(r) > r_max && r*u(1,1) > 0) {
        u = [0];
    }
} else { // 安定化制御
    u = -F * xheo(5:8,1);
}
return u;
}

Func Integer isSolverTrial(){
    return 0;
}

```