無キャスタ型対向二輪移動ロボットの開発 : 外力推定による安定化向上

末村, 昂輔 / SUEMURA, Kosuke

(発行年 / Year)
2009-03-24
(学位授与年月日 / Date of Granted)
2009-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2008 年度　修士論文

無キャスタ型対向二輪移動ロボットの開発
～外力推定による安定化向上～

Development of a Wheeled Mobile Robot without casters
- Improvement of Stabilization by Estimating External Forces -

指導教授　小林尚登

大学院工学研究科

電気工学専攻修士課程

学生証番号　07R3117

スエムラ　コウスケ

氏名　末村昂輔

Abstract

This research is about a stabilizing control of the moving robot, the mechanism of which is Wheeled Mobile Robot (WMR) without any casters to support its body. The main result shows an improvement of stabilization by using an observer that estimates external forces added to this mechanism. As well as computer simulation, we experimented in several situations. We found the control performance has been remarkably improved by compensating the estimated external force.

本論文の構成

本論文は，全体が５章から構成される．

第１章では，本研究の背景について述べる．第２章で提案する機構のシステムモデル，第３章では外力推定を用いた制御手法，さらに第４章ではシミュレーションと実験による検証を行う．そして最後に，第５章で結論を述べる．

目 次

図 目 次

第1章　はじめに

現在まで，左右それぞれにアクチュエータを取り付けた対向二輪移動ロボットが研究されている[1]~[3]．この機構は、一般的なステアリング機構と比較して，左右の車輪の回転をそれぞれ制御することにより，優れた旋回性を得ることができ，幅が狭く障害物の多い屋内における運搬ロボットとして活躍が見込まれている．

これらの対向二輪移動ロボットには，左右の駆動輪の他に，車体を支えるための補助輪があるが，その存在により，凹凸面に弱いことや，旋回性能に悪影響を及ぼすといった弱点がある．

そこで，本研究では対向二輪移動ロボットから，補助輪を排除した，無キャスタ型対向二輪移動ロボットの開発を目的とする．

補助輪を排除すると，車輪を駆動させたときの反力により車体が傾くので，車体の重心を考慮した制御によってその傾きを抑える必要がある．

現在，補助輪の無い対向二輪移動ロボットの研究としては，車輪型倒立振子の機構を扱ったものがほとんどである[4][5]．しかし，これらの移動ロボットは，回転軸に垂直な方向からの力を受けやすく，また，大きく傾いたときに人や物にぶつかる危険性が高いため，屋内での運搬ロボットとしては安全性の面で不安がある．

本研究での提案機構を Fig 1.1 に示す. 本機構の特徴は人に対する安全面を重要視し, 凸部がなく, 運搬物を内包できるようにしている. また, 補助輪を排除したことにより，車体がモータの反動により傾くといった弱点も，車体をドラム型としているので安全である.

しかし，車体をドラム型としたことで，運搬物の体積が車輪径により制限されてしまうので，体積の大きなものを運ぶためには車輪自体を大きくする必要がある．車輪径を大きくすることで，凹凸面での走行にさ

らに強くなるが，その一方で，慣性モーメントが増大し，粘性摩擦も低下するために，外部から力を受けた際に，素早く安定化することが特に重要となる．

そこで本論文では，無キャスタ型対向二輪移動ロボットの機構において，外力推定による安定化向上の有効性を，シミュレーションおよび実験によって示す．

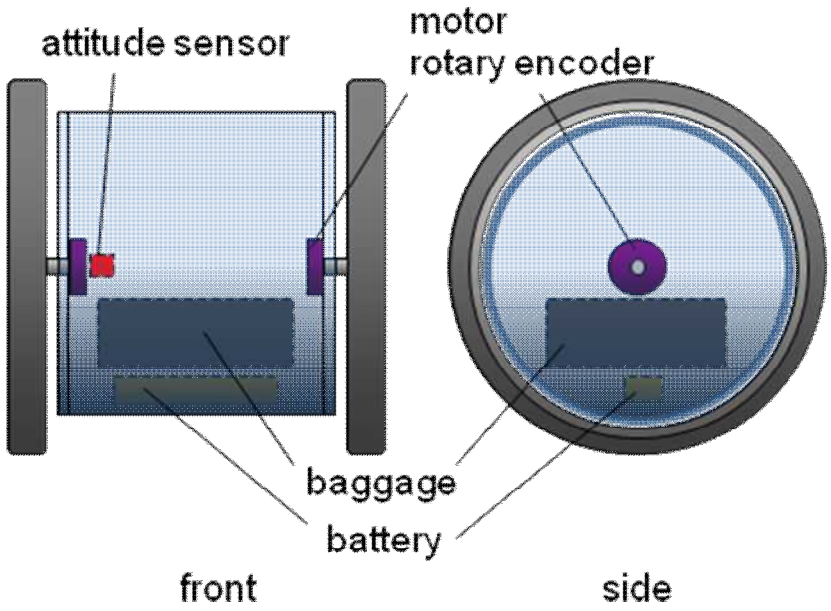


Fig 1.1: Structure of WMR without casters

第2章　無キャスタ型対向二輪移動ロボットのモデル化

本章では，無キャスタ型対向二輪移動ロボットについて，モデル化における前提事項を述べ，システムモデルを示す．そしてラグランジュの運動方程式の導出を行う．

2.1 前提事項

これまでに，車輪型倒立振子における外力オブザーバを用いた安定化制御法[6]が研究され，成果が挙げられている．

そこで本研究では，無キャスタ型対向二輪移動ロボットにその制御手法を応用し，ロボットに働く外力を車軸に加わる回転トルクと仮定してモデル化を行い，外力推定による安定化を試みる．

2.2 システムモデル

Fig 2.1 は無キャスタ型対向二輪移動ロボットを横から見たモデルである．θ_1，θ_2 はそれぞれ車輪の回転角，車体の傾斜角を示し，r は車輪半径，l は車軸と車体重心との距離を示している．

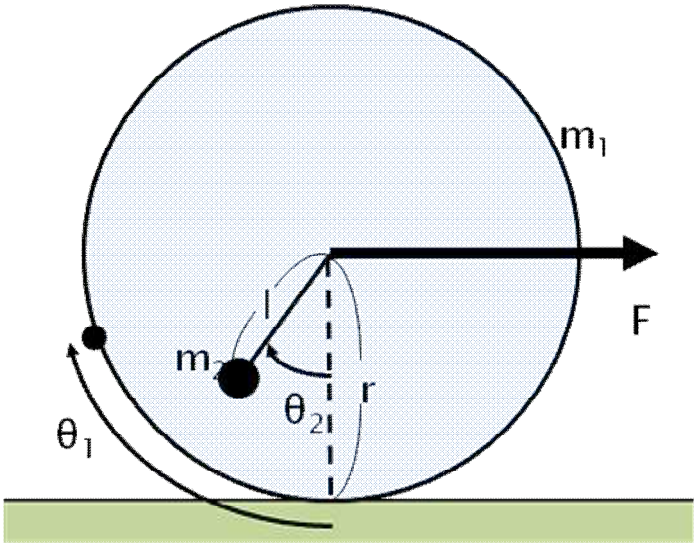


Fig 2.1: Model of WMR without casters

2.3 運動方程式の導出

Fig 2.1 のモデルにおいて，θ_1，θ_2 について，モータのトルク/電圧変換式（付録 第 A 章参照）を用いて，ラグランジュの運動方程式を以下のように導出した.

$$\begin{aligned}&\left(2J_1+(2m_1+m_2)r^2\right)\ddot{\theta}_1-m_2lr\cos\theta_2\cdot\ddot{\theta}_2\\&\quad+\left(2f_r+\frac{2K_\tau}{R}K_e\right)\left(\dot{\theta}_1-\dot{\theta}_2\right)\\&\quad+m_2lr\sin\theta_2\cdot\dot{\theta}_2{}^2=\frac{2K_\tau}{R}V_{in}+rF\end{aligned}\tag{2.1}$$

$$\begin{aligned}&-m_2lr\cos\theta_2\cdot\ddot{\theta}_1+\left(J_2+m_2l^2\right)\ddot{\theta}_2\\&\quad-\left(2f_r+\frac{2K_\tau}{R}K_e\right)\left(\dot{\theta}_1-\dot{\theta}_2\right)\\&\quad+m_2gl\sin\theta_2=-\frac{2K_\tau}{R}V_{in}+rF\end{aligned}\tag{2.2}$$

各パラメータについては Table 2.1 に示す.

式(2.1)，(2.2)について，車輪と車体はモータの回転に対し，作用反作用の関係でそれぞれ逆方向の力を受けるので，入力項の符号が異なっている．また，移動ロボットに働く外力を，車軸に加わる回転トルクと仮定したので，それぞれの外力項は，水平成分外力×車輪半径となっている.

Table 2.1: Parameters of the experimental system

θ_1	**車輪の回転角** [rad]	
θ_2	**車体の傾斜角** [rad]	
m_1	**車輪の質量** [kg]	0.273
m_2	**車体の質量** [kg]	2.776
l	**車軸と車体重心との距離** [m]	0.06
r	**車輪半径** [m]	0.15
g	**重力加速度** [m/ s^2]	9.8
J_1	**車輪の慣性モーメント** [kgm^2]	2.7
J_2	**車体の慣性モーメント** [kgm^2]	4.98
f_r	**駆動システムの粘性摩擦係数** [kg m^2/s]	2.0×10^{-4}
K_τ	**モータのトルク定数** [Nm/A]	2.1167
K_e	**モータ逆起電力定数** [Vs/rad]	3.30
R	**モータの電機子抵抗** [Ω]	8.633
V_{in}	**入力電圧** [V]	
F	**外力** [N]	

第3章　制御手法

本章では，まず第 2 章で求めた運動方程式から線形状態方程式を導出し，外力を推定するオブザーバを設計する．さらに，そのオブザーバにより求めた外力を用いた制御系を構成する．

3.1 オブザーバによる外力推定

式(2.1)，(2.2)について，平衡点近傍で線形化すると以下の運動方程式が得られる．

$$\begin{aligned}&\left(2J_1+(2m_1+m_2)r^2\right)\ddot{\theta}_1-m_2lr\ddot{\theta}_2\\&\qquad+\left(2f_r+\frac{2K_\tau}{R}K_e\right)\left(\dot{\theta}_1-\dot{\theta}_2\right)\\&\qquad=\frac{2K_\tau}{R}V_{in}+rF\end{aligned}\tag{3.1}$$

$$\begin{aligned}&-m_2lr\ddot{\theta}_1+\left(J_2+m_2l^2\right)\ddot{\theta}_2\\&\qquad-\left(2f_r+\frac{2K_\tau}{R}K_e\right)\left(\dot{\theta}_1-\dot{\theta}_2\right)\\&\qquad+m_2gl\theta_2=-\frac{2K_\tau}{R}V_{in}+rF\end{aligned}\tag{3.2}$$

移動ロボットに働く外力 F を，各角度，各角速度に加えて，状態量の一つとしてシステムに組み込みオブザーバによる推定を行う．

外力 F はステップ状に変化し，そのダイナミクスを

$$\dot{F}=0\tag{3.3}$$

と仮定すると，4 次のシステムを拡張して，5 次のシステムとして次のように状態方程式と出力方程式が表せる．

$$\begin{cases}\dot{x}=Ax+bV_{in}\\y=Cx\end{cases}\tag{3.4}$$

ただし

$$x = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ F \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & a_1 & a_3 & -a_3 & a_5 \\ 0 & a_2 & a_4 & -a_4 & a_6 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ 0 \\ b_1 \\ b_2 \\ 0 \end{bmatrix}$$

$$y = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$a_1 = -gm_2{}^2 l^2 r/\Delta$$
$$a_2 = -gm_2 l\left(2J_1 + (2m_1 + m_2)r^2\right)/\Delta$$
$$a_3 = -2\left(J_2 + m_2 l(l-r)\right)\left(K_e K_\tau + f_r R\right)/R\Delta$$
$$a_4 = -2\left(lm_2 r - 2J_1 - (2m_1 + m_2)r^2\right)\left(K_e K_\tau + f_r R\right)/R\Delta$$
$$a_5 = r\left(J_2 + m_2 l(l+r)\right)/\Delta$$
$$a_6 = r\left(2J_1 + r(2m_1 r + m_2(l+r))\right)/\Delta$$
$$b_1 = 2K_\tau\left(J_2 + m_2 l(l-r)\right)/\Delta R$$
$$b_2 = -2K_\tau\left(2J_1 + r(-m_2 l + (2m_1 + m_2)r)\right)/\Delta R$$
$$\Delta = 2J_1\left(J_2 + m_2 l^2\right) + \left(2m_1 m_2 l^2 + J_2(2m_1 + m_2)\right)r^2$$

このシステムで直接観測できる状態量は θ_1 と θ_2 のみであるため，未知の各角速度 $\dot{\theta}_1$，$\dot{\theta}_2$ と外力 F について，オブザーバによる推定が必要である．

このシステムは可観測であり，オブザーバの構成が可能である．従って，未知の状態量 $\dot{\theta}_1$，$\dot{\theta}_2$，F をオブザーバにより推定することが可能である．移動ロボットに与える操作入力と，その操作入力のときの出力をオブザーバに入力することにより全状態量を推定する．これを式で表すと次のようになる．

$$\dot{x}^* = (A - KC)x^* + Ky + bV_{in} \tag{3.5}$$

ただし

$$x^* = \begin{bmatrix} \theta_1^* & \theta_2^* & \dot{\theta}_1^* & \dot{\theta}_2^* & F^* \end{bmatrix}^T$$

θ_1^*～F^*はオブザーバによって推定された状態変数である．また，K はオブザーバゲイン行列であり，A－KC の極を任意に指定でき，それにより推定値を真値に追従させることが可能である．

この構成をブロック線図で示すと，Fig 3.1 のようになる．

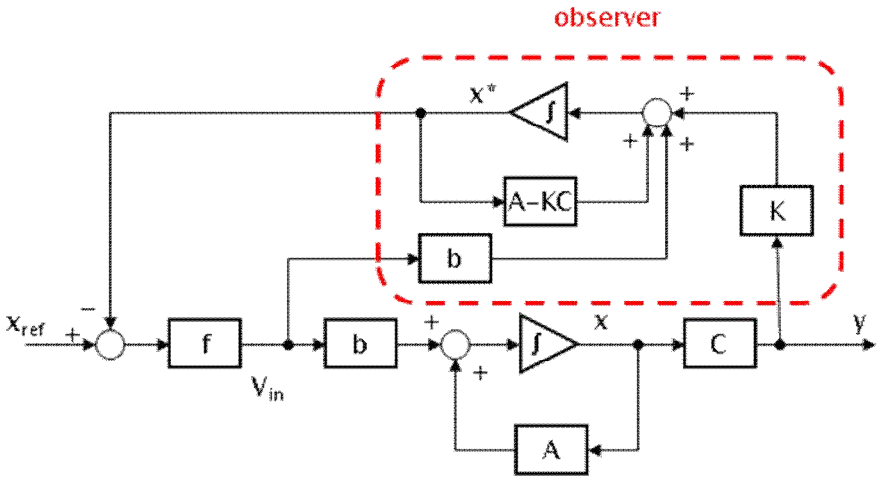


Fig 3.1: Block diagram of control system

3.2 外力補償項の追加

平衡状態では運動方程式(2.1)，(2.2)は次のようになる．

$$0 = \frac{2K_\tau}{R} V_{off} + rF \tag{3.6}$$

$$m_2 gl \sin\theta_{2b} = -\frac{2K_\tau}{R} V_{off} + rF \tag{3.7}$$

ここで，V_{off} は平衡状態保持電圧で，式(3.6)，(3.7)は外力 F が働くとき，入力を $V_{off} = -rFR/2K_\tau$ とすると車体が θ_{2b} 傾いた状態で安定することを示す．

つまり，外力が働くときの制御系は，外力が働かない場合の制御系に，外力によって傾いた車体傾斜を保持するための，補償電圧 V_{off} をフィードフォワード項として加えたものとなる．そのときの制御則は次のようになる．

$$V = -f \begin{bmatrix} \theta_1^* - \theta_{1ref} \\ \theta_2^* - \theta_{2b}^* \\ \dot{\theta}_1^* - \omega_{1ref} \\ \dot{\theta}_2^* \end{bmatrix} + V_{off}^* \tag{3.8}$$

ただし，

$$\theta_{2b}^* = \frac{2rF^*}{m_2 gl}$$

$$V_{off}^* = -\frac{R}{2K_\tau} rF^*$$

$$f = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix}$$

ここで，θ_{1ref}，ω_{1ref}はそれぞれ，目標車輪回転角度，角速度である．また，f は外力が働かない場合の，式(3.4)から F を除いた４次のシステムのフィードバックゲインベクトルである．

さらに，式(3.8)は次のように表せる．

$$V = -f\begin{bmatrix} \theta_1^* - \theta_{1ref} \\ \theta_2^* \\ \dot{\theta}_1^* - \omega_{1ref} \\ \dot{\theta}_2^* \end{bmatrix} - \left(\frac{R}{2K_\tau} - \frac{2f_2}{m_2 gl} \right) rF^* \tag{3.9}$$

ここで，

$$f_5 = \left(\frac{R}{2K_\tau} - \frac{2f_2}{m_2 gl} \right) r \tag{3.10}$$

とすると制御則は，

$$\tau = -f'\left(x^* - x_{ref} \right) \tag{3.11}$$

と表せる．ただし，

$$f = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 & f_5 \end{bmatrix}$$
$$x_{ref} = \begin{bmatrix} x_{1ref} & 0 & \omega_{1ref} & 0 & 0 \end{bmatrix}^T$$

第4章　シミュレーションと実験

本章では，第 3 章で導出した制御則による，シミュレーションと実験の結果を提示し，考察を行う．

シミュレーションと実験は，常に外力を受ける状態での制御として傾斜面での走行と，突発的に外力が働いたときの安定化の 2 種類の状況を想定し，それぞれ外力補償がある場合(case1)と，ない場合(case2)との比較を行った．

本実験に用いた試作機の構成については付録 第 B 章に記載する．

4.1　傾斜面での走行

傾斜面では，機体の重量と傾斜角により決まる，常に一定の外力を受けることになる．そこで，ここでは傾斜面での上り走行において，前章で示した外力推定を用いた制御をシミュレーションと実験を通して行う．

屋内におけるスロープの勾配は，国土交通省が定めたバリアフリー新法により，1/12（　4.76[°]）以下と基準が定められている．そこで，今回は勾配 1/12 のスロープにおける，上り走行のシミュレーションと実験を行った．なお，このスロープで移動ロボットが受ける外力 F は上りにおいて$-(2m_1+m_2)\, g \sin(0.08314)$ [N]である．

移動ロボットの動作は停止状態から徐々に加速し，1.2[m]進んだ位置で停止する．その際の制御は，目標車輪回転角 θ_{1ref} を 0〜8[rad]まで徐々に変化させ，それを追従させることで行う．

ここで用いるオブザーバゲインは，サンプリングタイム 10[ms]において発散せず，かつ外力が速やかに収束する値を，フィードバックゲインは追従性と車体傾斜角の安定性の良いものを，それぞれ最適レギュレータ法（付録 第 C 章参照）から求め，シミュレーションにより選定した．それぞれの最適レギュレータ法の評価関数におけるパラメータは，オブザーバゲインについては Q=diag(1, 1, 1, 1, 10^4)，R=diag(10, 10)，フィードバックゲインについては Q=diag(4, 240, 1, 1)，R=1 である．

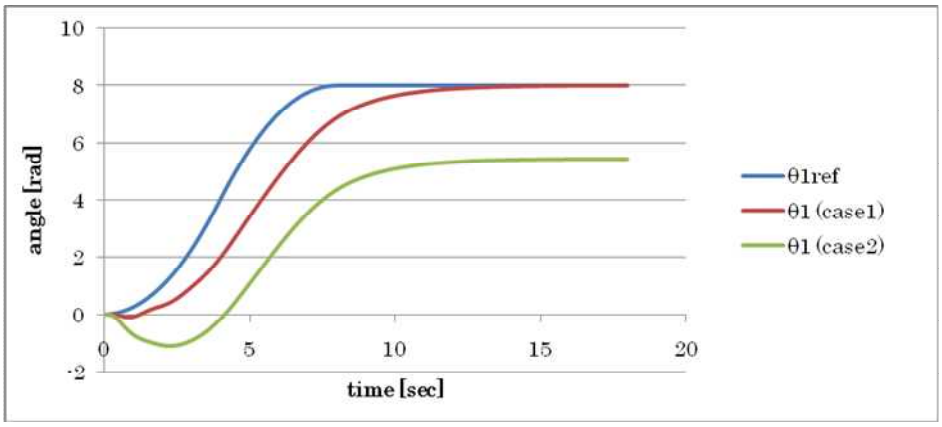
実験の様子を Fig 4.1 に，シミュレーションと実験の結果をそれぞれ Fig 4.2，Fig4.3 に示す．

シミュレーションと実験結果を比較すると，実験結果の推定外力は，シミュレーションに比べて実際の外力に収束するまでに長い時間を要した．これは加速度センサに車体の振動や，僅かな進行加速度によってノイズが乗ったために，オブザーバの収束が遅くなったためだと考えられる．

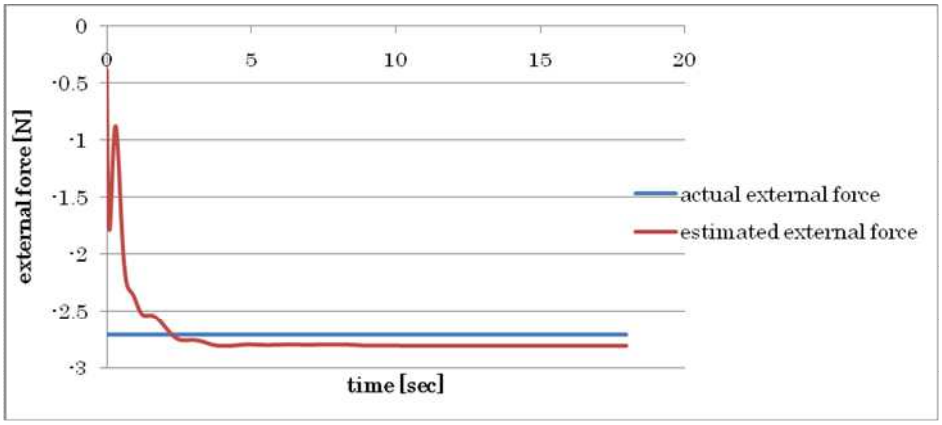
また，シミュレーションと実験結果で，ともに外力補償がない場合，車輪回転角が目標値に収束しなかった．これは入力に外力補償項が存在しないために，車輪回転角が目標値とずれた値で，車体傾斜角の項と釣

りあったためである．このことから傾斜面など常に外力が加わる状況において，外力を観測し補償することが，正確に目標とする位置に停止するために必須であることが分かる．

Fig 4.1: Experiment scene about movement on slope

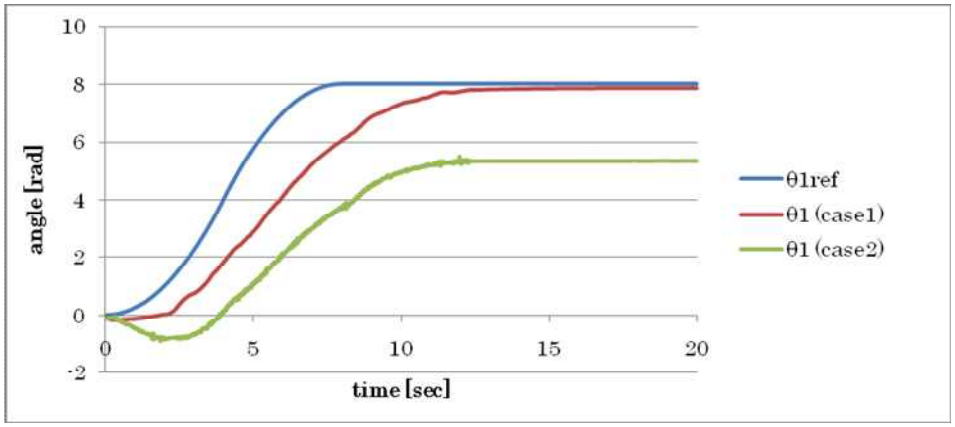


(a) angle of wheels

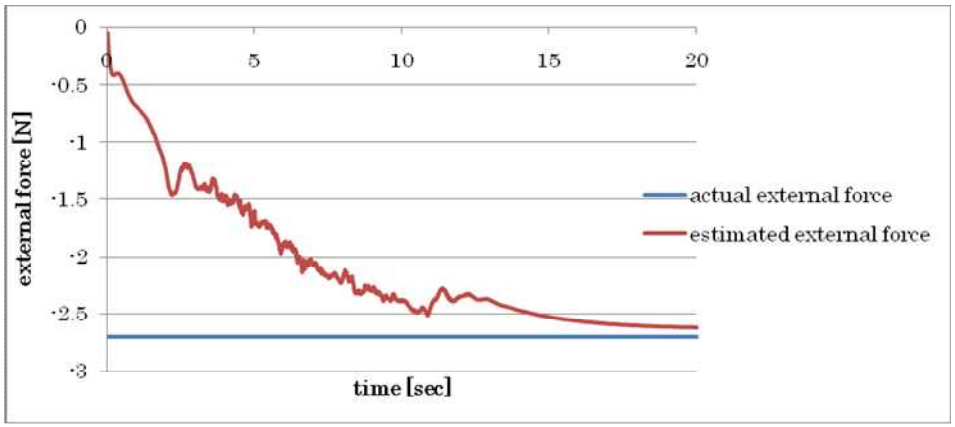


(b) external force

Fig 4.2: Simulation result on slope



(a) angle of wheels



(b) exteranal force

Fig 4.3: Experiment result on slope

4.2 突発的外力に対する安定化

水平面において，時刻 t = 6〜11 [s]の間に，F = -2.94 [N]のステップ状の外力を与えたときの，シミュレーションと実験を行った．

実験装置を Fig 4.4 に示す．車輪の回転軸に糸を通し，滑車を使用して 300g の重りが吊るされている．実験方法は，重りを持ち上げた状態から始めて，時刻 t=6[s]で重りを吊り下げ，t=11[s]で再び重りを持ち上げることで，ステップ状の外力を与える．

オブザーバゲインは傾斜面での走行実験と同じものを使用し，フィードバックゲインは最適レギュレータ法を用いて，車輪回転角の小ささを優先し，シミュレーションにより選定した．最適レギュレータ法の評価関数のパラメータは，Q=diag(4, 20, 1, 1)，R=1 である．

実験の様子を Fig 4.5 に，シミュレーションと実験の結果をそれぞれ Fig 4.6，Fig4.7 に示す．

シミュレーションと実験結果を比較すると，実験結果の方が車輪の回転角度がシミュレーションの値より小さいことが分かる．これはモータの電機子反作用による回転の抑制や，駆動システムの粘性摩擦が想定した値より大きかったことが原因と考えられる．また，推定外力の値が小さいことについても，この回転角が小さかったことに起因すると考えられる．

しかし，シミュレーションと実験結果のいずれも，外力補償をした場合が，補償しなかった場合に比べ，車輪の回転角が小さくなっていることが分かる．安全面を重視する場合，外力に対して車体位置の変動が小さいことが望ましいので，突発的な外力に対しても外力を観測し補償することが有効であることが言える．

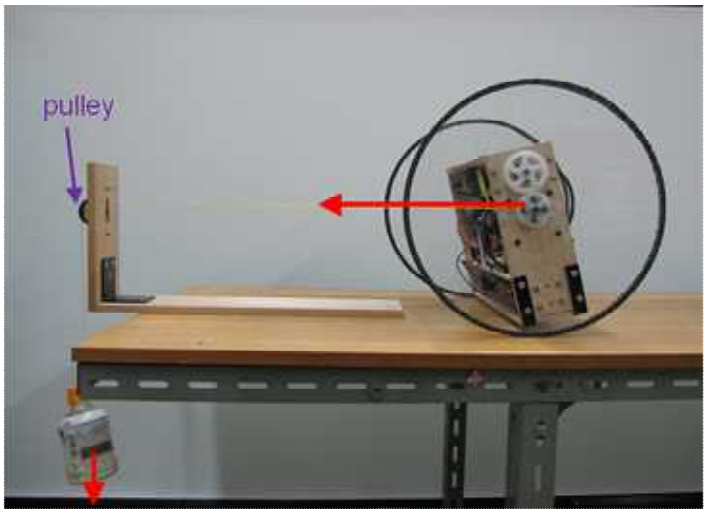
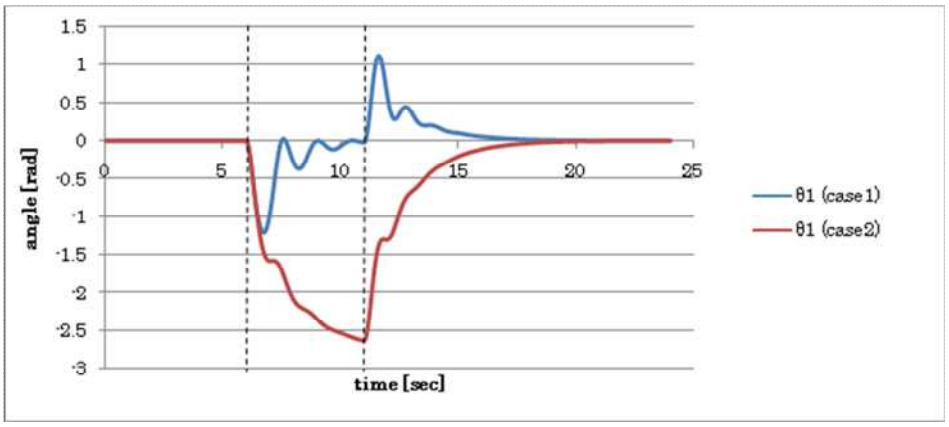
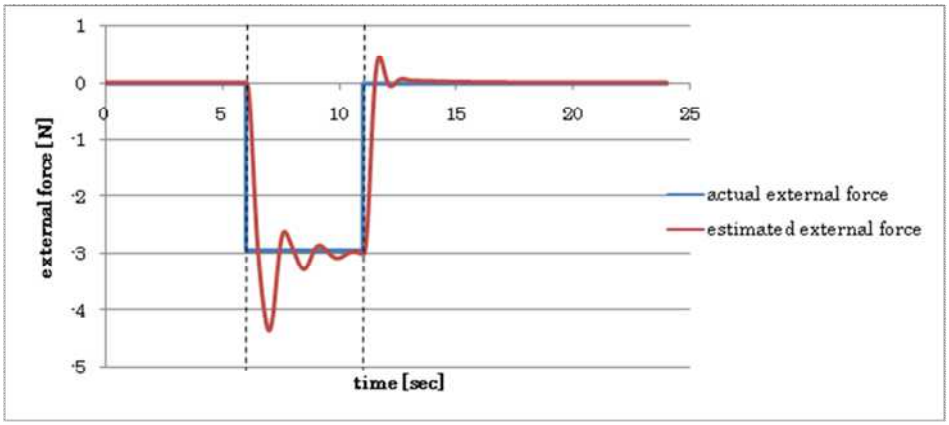


Fig 4.4: Experiment equipment about stabilization with disturbance

Fig 4.5: Experiment scene about stabilization with disturbance

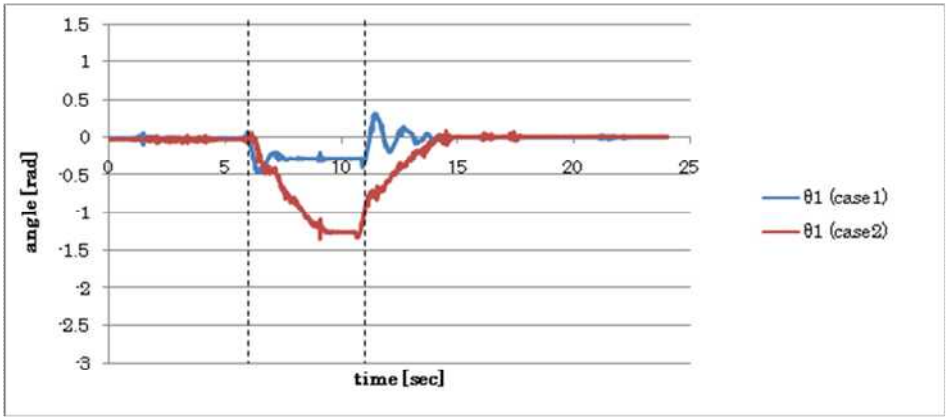


(a) angle of wheels

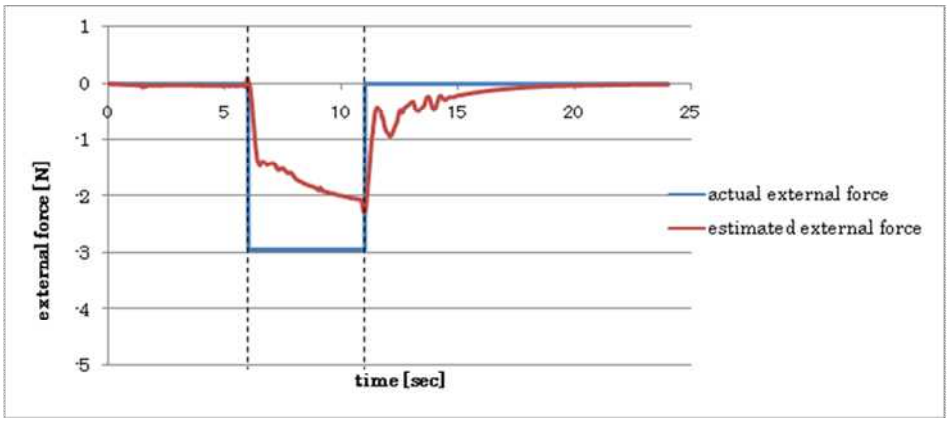


(b) external force

Fig 4.6: Simulation result with disturbance



(a) angle of wheels



(b) external force

Fig 4.7: Experiment result with disturbance

第5章　おわりに

本稿では，無キャスタ型対向二輪移動ロボットにおける，外力推定を用いた，定常外力が加わるときと，突発的外力が加わったときの，2 つの状況においてのシミュレーションおよび実験を行い，それぞれについてその有効性を示した．これにより，無キャスタ型対向二輪車が機動性に優れ，かつ安定性を確保した移動ロボットとして期待できる結果となった．

なお，今回製作した試作機は，車体の傾斜角の測定に加速度センサを使用したために，ノイズの影響によりオブザーバの収束性が悪くなるといった問題があったが，その解消法としてカルマンフィルタを設計することや，松本らのレートジャイロとオブザーバによる傾斜角の測定法[7]を使用するといったことが考えられる．

また，今回は直進運動のみでの制御を行ったが，今後は旋回運動における制御などが課題として挙げられる．

謝辞

まず，指導教授である小林尚登教授に感謝の意を申し上げます．小林教授には，知識や経験の乏しい私に，様々な助言をして頂き，多くのご指導を承りました．

理化学研究所の川端先生には，正規の仕事でご多忙であるにも関わらず，われわれの研究について親身になって相談に乗って頂きありがとうございました．

また，同研究室の先輩、同輩、後輩の皆様に，研究や学業をサポートして頂いたことに感謝します．

最後に，私の大学生活を様々な面で支援して頂いた両親に感謝の意を示し，本論文の結びとさせていただきます．

参考文献

[1] 水谷，大淵，対馬：“ 自律移動車輪型ロボットの実際位置に関する軌道追従制御 （第 1 報,レーザ燈台測定システムによる推定姿勢角度の修正を行った場合の軌道追従制御特性）”, 日本機械学會論文集（C 編）61 巻，583 号，pp1084〜1089，1995

[2] 関，高，神谷，疋津，張：“ 繰り返し順変換を用いた独立二輪駆動型移動ロボットにおける移動経路の生成 ”, 精密工学会誌論文集 Vol.71 No.4，pp506〜511，2005

[3] 小林，對馬：“ 位置誤差を考慮した移動ロボットの軌道追従制御（ベジェ曲線による軌道補正法）”，ロボティクス・メカトロニクス講演会’02 講演論文集

[4] 尾崎，大串，下川，林：“ 車輪型倒立振子の位置・姿勢制御 ” 日本機械学会論文集（C 編） Vol.65 No.637，pp3635〜3642，1999

[5] 中代, 古谷, :“ 倒立振子を応用した車輪型移動ロボットに関する研究 ”, 日本機械学会関東支部総会講演会講演論文集 2002(8)，pp383-384

[6] 城間，松本，谷:“ 構造的に不安定な移動ロボットの協調行動による物体の運搬 ”, 日本機械学会論文集(C 編)64 巻 628 号，pp164〜171，1998

[7] 松本，梶田，谷：“ 移動ロボットの内界センサのみによる姿勢検出とその制御 ”, 日本ロボット学会誌 Vol.8 No.5，pp37〜46，1990

[8] 吉井恒夫，井村順一：“ 現代制御理論 ”, 昭晃社，1994

第A章　トルク/電圧変換式

Fig A.1 の電機子回路について, キルヒホッフの電圧則から次式が成り立つ.

$$L\frac{d}{dt}i + Ri + e = V$$

回路のインダクタンス L は十分に小さいと仮定し, 電圧 V を制御入力とすると，電流 i は次式で表わされる.

$$i = \frac{V - e}{R}$$

モータにより発生するトルクτ，回転速度ω，逆起電力 e とすると

$$e = K_e\omega$$
$$\tau = K_\tau i = K_\tau \frac{V - e}{R} = K_\tau \frac{V - K_e\omega}{R}$$

となる．このとき K_τ はトルク定数であり，K_e は逆起電力定数である.

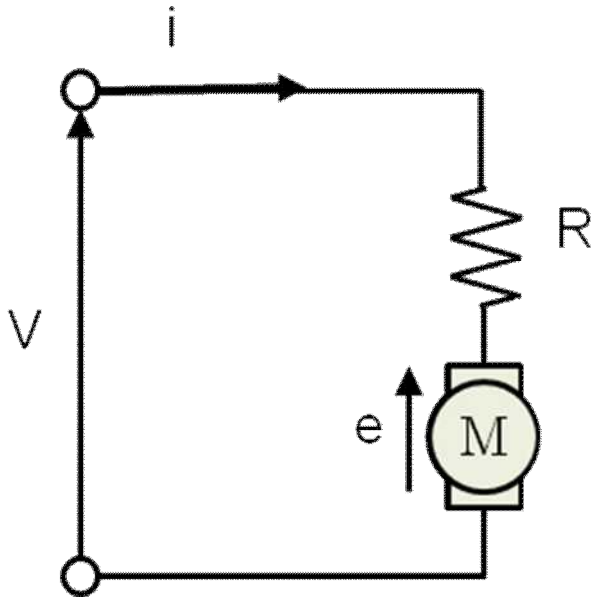


Fig A.1: Motor circuit

第B章　試作機の構成

今回の実験に使用した試作機を Fig B.1 に示す．試作機の各パラメータは第 2 章の Table 2.1 と同じ値である．なお，本試作機は外力推定の実験を目的として製作したため，実験の行い易さを優先し，Fig 1.1 のように車体をドラム型にはしていない．

本試作機の構成要素を以下に示す．

(1) **モータ**
日本サーボ(株)製の DME33B6HPA に同社のギヤヘッド 6DG150 を取り付けたもの(Fig B.2)を 2 つ使用した．仕様を Table B.1 に示す．

(2) **ロータリーエンコーダ**
マイクロテックラボラトリー(株)製の MES-30-1500 (Fig B.3)を使用した．仕様を Table B.2 に示す．なお，モータ側に 40 歯，ロータリーエンコーダ側に 50 歯のギヤで接続されているので，車輪角度一周あたり 1200 パルスとなる．

(3) **3 軸加速度センサ**
車体の傾斜角測定にはカイオニクス社の 3 軸加速度センサモジュール KXM52-1050 (Fig B.4)を使用し，重力加速度の方向から傾斜角を測定する．仕様を Table B.3 に示す．

(4) **バッテリ**
Kung Long Batteries industrial 社製の WP-2.6-12 完全密封型鉛畜電池（12V 2.6Ah）(Fig B.5)を使用した．

(5) **コントロールユニット**
制御回路を Fig B.6 に示す. 制御用マイコンには PIC18F452 を使

用する. 加速度センサからの信号は作動増幅回路により 0〜5[V] の範囲に増幅され，MCP3208 により 12 ビット AD 変換され，PIC18F452 に送られる．このとき PIC では取得値について 10 回毎の平滑化を行う．また，ロータリーエンコーダからの信号は PIC16F873 で計測され，PIC18F452 に送られる．モータの制御にはモータドライバ TA8440HQ を使用し，PIC18F452 からの PWM 信号により制御する．

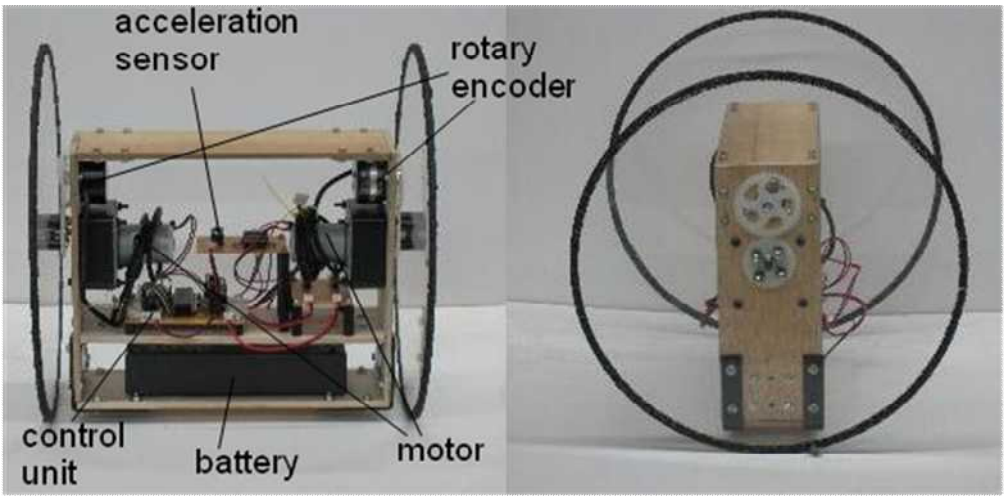


Fig B.1: Prototype in this reseach

Fig B.2: Motor

Table B.1: Spec of motor

定格出力	3 [W]
定格電圧	DC 12 [V]
減速比分母	150
回転速度	37 [r/min]
許容トルク	0.77 [N•m]
質量	400 [g]

Fig B.3: Rotary encoder

Table B.2: Spec of rotary encoder

定格電圧	DC　5～12 [V]
検出方式	**インクリメンタル**
出力パルス数	1500 [**パルス/回転**]
出力相	A，B，Z **相**
出力形態	**矩形波**
最高応答周波数	100 [kHz]

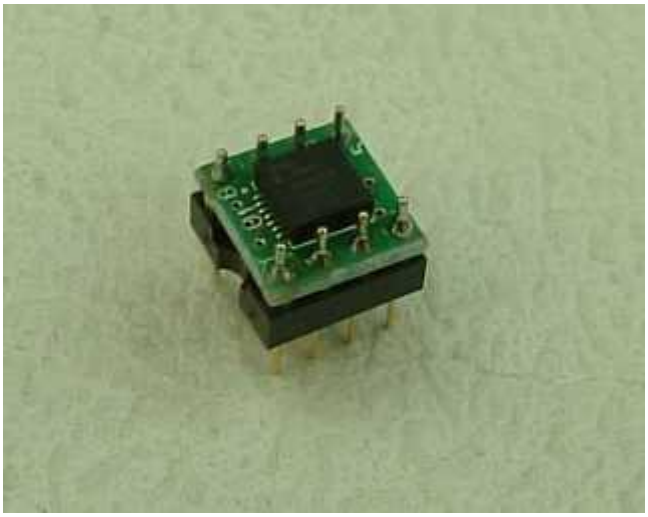


Fig B.4: Tri-axis accelerometer module

Table B.3: Spec of tri-axis accelerometer module

定格電圧	2.5～5.5 [V]	
軸	3(xyz)	
検出範囲	±2.0 [g]	
出力	660 [mV/g]	入力 3.3 [V]時
オフセット電圧	1.65 [V]	入力 3.3 [V]時
出力周波数	0～1500 [Hz] (xy) 0～1500 [Hz] (z)	

Fig B.4: Battery

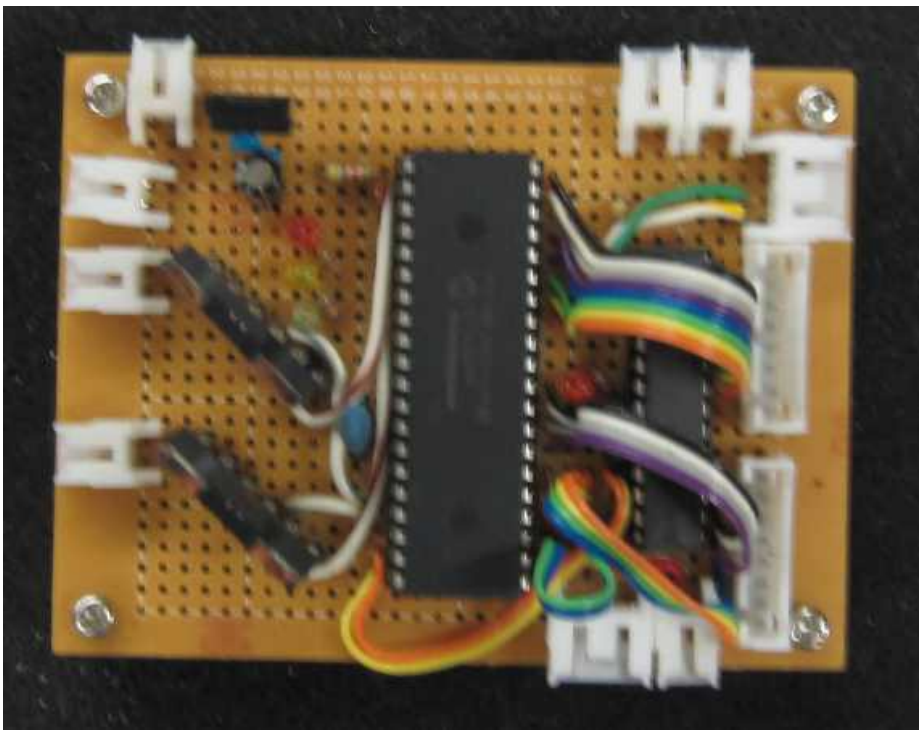


Fig B.5: Control circuit

第C章　最適レギュレータ法

可制御である線形システム

$$\dot{x} = Ax + Bu;\ x \in \Re^n,\ u \in \Re^r$$

において二次評価関数

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt;\ R \in \Re^{r \times r},\ Q \in \Re^{n \times n}$$

を最小にする最適制御入力は

$$u = Kx,\ K = -R^{-1} B^T P$$

の状態フィードバック制御則である．ここで $P \in \Re^{n \times n}$ **は** Riccati **の行列方程式**

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

を満たす唯一の正定解である．ここで R，Q **は正定行列とする．**

第D章　シミュレーションプログラム

第４章で行ったシミュレーションのＣ言語プログラムを記載する．このプログラムはOpenGLを使用し，3Dグラフィックス化をしている．

```
#define WIN32_LEAN_AND_MEAN                    // Windows ヘッダーから使用されていない部分を除外

// Windows ヘッダー ファイル
#include <windows.h>
#include <iostream.h>
#include<stdio.h>
#include<math.h>
#include <conio.h>
#include <time.h>

#include <GL/gl.h>
#include <GL/glut.h>

// C ランタイム ヘッダー ファイル
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
```

```
#include <tchar.h>

#define pi   3.14159265358979323846
#define g    9.8

void ErrorHandler(char *message, DWORD error);
void move();

double euler(double X[5], double OX[5], double OG[5][2], double A[5][5], double B[5]);
double runge_kutta(double X[5]);
double observer(double X[5], double OX[5], double OG[5][2], double A[5][5], double B[5]);

void Matrix_addition(double A[5], double B[5], double C[5], double d);
void Matrix_55(double A[5][5], double B[5], double C[5]);
void F(double X[5], double f[2]);
void V(double X[5], double v[2]);
void OY(double X[5], double OX[5], double OG[5][2], double oy[5]);

static int width = 640, height = 240;                    // ウィンドウサイズ

static double draw_count = 0;

double i, j;
static long t = 0;
static long field = 10;                                  // フィールド範囲

static double alpha = 0.08314;                           // 床傾斜角

static double r = 0.15;                                  // タイヤ半径
static double th1 = 0;                                   // タイヤ角度
static double dth1 = 0;                                  // タイヤ角速度
```

```
static double l = 0.06;                          // 車体中心から重心までの距離
static double th2 = 0;                           // 重心角度
static double dth2 = 0;                          // 重心角速度

static double phi = 0.0;                         // 進行方向角度

static double dt = 0.01;                         // サンプリングタイム

static double wid = 0.15;                        // 車幅
static double x = 0.0, y = 0.0, z = 0.0;         // 車体中心座標

static double Px, Py;                            // 視点座標

static double m1 = 0.273;                        // 車輪質量
static double m2 = 2.776;                        // 車体質量

static double fr = 0.0002;                       // モータの粘性摩擦
static double J1, J2;                            // 車輪、車体の慣性モーメント
static double Kt = 2.1167;                       // モータのトルク係数
static double Ke = 3.3;                          // モータの逆起電力定数
static double R = 8.633;                         // モータの内部抵抗

static double Vin;                               // 入力電圧

static double tau;                               // 入力トルク

static double fo = 0;                            // 外力

static double th1ref, dth1ref;                   // 目標値

FILE *fp;
```

```
GLfloat mat_shininess[] = {1.0};                    // 光量
GLfloat light_position[] = {0.0, -10.0, 0.0, 1.0};  // 光源位置

void color(double red, double green, double blue)
{
        GLfloat mat_ambient[] = {red, green, blue, 1.0};
        GLfloat mat_diffuse[] = {red, green, blue, 1.0};
        GLfloat mat_specular[] = {red, green, blue, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
}

void init(void)
{
        glClearColor(1.0, 1.0, 1.0, 0.0);
        glShadeModel(GL_SMOOTH);
        color(1.0, 1.0, 1.0);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
}

void cube(int mode, double x, double y, double z)
{
        glPushMatrix();
                if(mode == 1){glTranslated(x/2, 0.0, 0.0);}
                if(mode == 2){glTranslated(0.0, y/2, 0.0);}
                if(mode == 3){glTranslated(0.0, 0.0, z/2);}
```

```
                glScaled(x, y, z);
                glutSolidCube(1.0);
        glPopMatrix();
}

void wire_sphere(double x, double y, double z)
{
        glPushMatrix();
                glScaled(x, y, z);
                glutWireSphere(1, 12, 2);
        glPopMatrix();
}

/*****************************************************************************
*                              オブジェクトの設定
*****************************************************************************/

void clip()
{
        glScalef(3, 3, 3);
/*  地面  */
        /* y 軸方向  */
        color(0.0, 1.0, 1.0);
        glLineWidth(0.02);
        glBegin(GL_LINES);
            glColor3d(0.0, 1.0, 0.0);
                for(i = 0; i <= field; i += 0.5){
                glVertex3f((field/2 - i)*cos(alpha), field/2, (field/2 - i)*sin(alpha));
                glVertex3f((field/2 - i)*cos(alpha), -field/2, (field/2 - i)*sin(alpha));
                }
        glEnd();
```

```
        /* x 軸方向 */
        color(0.0, 1.0, 1.0);
        glBegin(GL_LINES);
            glColor3d(0.0, 1.0, 0.0);
                for(i = 0; i <= field; i += 0.5){
                glVertex3f(field/2*cos(alpha), field/2 - i, field/2*sin(alpha));
                glVertex3f(-field/2*cos(alpha), field/2 - i, -field/2*sin(alpha));
                }
        glEnd();

/*  車体 */
        glPushMatrix();
                glTranslated(x, y, r+z);

        glPushMatrix();
                color(0.0, 1.0, 0.0);
                glRotated(180, 0.0, 1.0, 0.0);
                glRotated(th2/pi*180, 0.0, 1.0, 0.0);
                cube(3, 0.02, wid, 0.12);
        glPopMatrix();

/*  車輪 */
        glPushMatrix();
                color(0.0, 0.0, 0.0);
                glTranslated(0.0, 0.5*wid, 0.0);
                glRotated(90, 1, 0, 0);
                glRotated(-th1*180/pi, 0, 0, 1);
                glutWireTorus(0.01, r, 20, 50);
                wire_sphere(r, r, 0.01);
        glPopMatrix();

        glPushMatrix();
```

```
                color(0.0, 0.0, 0.0);
                glTranslated(0.0, -0.5*wid, 0.0);
                glRotated(90, 1, 0, 0);
                glRotated(-th1*180/pi, 0, 0, 1);
                glutWireTorus(0.01, r, 20, 50);
                wire_sphere(r, r, 0.01);
        glPopMatrix();

        glPopMatrix();

/*  目標値  */
        glPushMatrix();
                glTranslated(th1ref*r*cos(alpha), 0, 0.15+th1ref*r*sin(alpha));
                color(1.0, 0.0, 0.0);
                glScaled(1, 1, 1);
                glutSolidCube(0.02);
        glPopMatrix();

}

void draw(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
                glColor3d(0.0, 0.0, 0.0);
                clip();
        glPopMatrix();
        glutSwapBuffers();
        glFlush();
}

void reshape(int width, int height){
```

```
        glViewport(0, 0, (GLsizei)width, (GLsizei)height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(60.0, (GLfloat)width / (GLfloat)height, 1.0, 200.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0, -1.6, 0.7, 0, 0.3, 0, 0.0, 0, 1);
}


/*****************************************************************************
*                              目標値と外力の設定
*****************************************************************************/

void target(){
//  傾斜面走行

        if(t*dt < 4){
                th1ref = 0.25*t*dt*t*dt;
                dth1ref = 0.5*t*dt;
        }
        else if(t*dt < 8){
                th1ref = 8-0.25*(8-t*dt)*(8-t*dt);
                dth1ref = 0.5*(8-t*dt);
        }
        else{
                th1ref = 8;
                dth1ref = 0;
        }

        fo = -(2*m1+m2)*g*sin(alpha);
```

```
/*
//   突発的外力

            if(t*dt < 6){fo = 0;}
            else if(t*dt < 11){fo = -0.3*g;}
            else{fo = 0;}
            alpha = 0;
*/

}

/*************************************************************************
*                                制御部
*************************************************************************/

void move(void){

            static double A[5][5];                  // システム行列
            static double B[5];

            static double FG[5];                    // フィードバックゲイン
            static double OG[5][2];                 // オブザーバゲイン

            static double X[5];                     // θ行列
            static double OX[5];                    // 推定θ行列
            static double Xref[5];                  // 目標θ行列

            J1 = m1*r*r/2;                          // 車輪慣性モーメント
            J2 = m2*l*l*4/3;                        // 車体慣性モーメント

            target();
```

```
X[0] = th1;
X[1] = th2;
X[2] = dth1;
X[3] = dth2;
X[4] = fo;

Xref[0] = th1ref;
Xref[1] = 0;
Xref[2] = dth1ref;
Xref[3] = 0;
Xref[4] = 0;

// システム行列
A[0][0] = 0;   A[0][1] = 0;   A[0][2] = 1;   A[0][3] = 0;   A[0][4] = 0;
A[1][0] = 0;   A[1][1] = 0;   A[1][2] = 0;   A[1][3] = 1;   A[1][4] = 0;
A[2][0] = 0;
A[2][1] = -32.3155;
A[2][2] = 2.1364;
A[2][3] = -2.1364;
A[2][4] = 5.7413;
A[3][0] = 0;
A[3][1] = -104.6240;
A[3][2] = 71.7034;
A[3][3] = -71.7034;
A[3][4] = 12.5841;
A[4][0] = 0;   A[4][1] = 0;   A[4][2] = 0;   A[4][3] = 0;      A[4][4] = 0;

B[0] = 0;
B[1] = 0;
B[2] = -0.6472;
B[3] = -21.7229;
B[4] = 0;
```

```
// オブザーバゲイン
// QQ=diag([1,1,1,1,10000]);, rr=diag([10,10]);
        OG[0][0]= 4.3868;      OG[0][1]= 4.2405;
        OG[1][0]= 4.2405;      OG[1][1]= 4.5963;
        OG[2][0]=18.5630;      OG[2][1]=21.6434;
        OG[3][0]=16.4498;      OG[3][1]=19.5042;
        OG[4][0]=20.7915;      OG[4][1]=23.8267;

// オブザーバによる推定
        OX[5] = observer(X, OX, OG, A, B);

// フィードバックゲイン
        FG[0] = 2.0000; FG[1] = -16.3227; FG[2] = 0.5451; FG[3] = -0.3850;
                //Q=diag([4,240,1,1]);, r=1;  傾斜面走行
//      FG[0] = 2.0000; FG[1] = -8.3415; FG[2] = 0.4003; FG[3] = -0.2763;
                //Q=diag([4,20,1,1]);, r=1;  突発的外力

        FG[4] = - (2*FG[1]/(m2*g*l) - R/(2*Kt))*r;

// 外力補償項あり
        Vin = FG[0]*(Xref[0] - OX[0]) + FG[1]*(Xref[1] - OX[1]) + FG[2]*(Xref[2] - OX[2]) +
FG[3]*(Xref[3] - OX[3]) + FG[4]*(Xref[4] - OX[4]);

// 外力補償項なし
//      Vin = FG[0]*(Xref[0] - OX[0]) + FG[1]*(Xref[1] - OX[1]) + FG[2]*(Xref[2] - OX[2]) +
FG[3]*(Xref[3] - OX[3]);

// ルンゲクッタ法による出力の導出
        X[5] = runge_kutta(X);

        th1 = X[0];
```

```
		th2 = X[1];
		dth1 = X[2];
		dth2 = X[3];

		x = r * th1 * cos(alpha);
		z = r * th1 * sin(alpha);

		tau = Kt*(Vin - Ke*(X[2]-X[3]))/R;

		printf("%lf %lf %lf %lf %lf %lf %lf %lf
		%lf %lf %lf %lf %lf¥n", t*dt, th1ref, X[0], OX[0], X[1],
OX[1], X[2], OX[2], X[3], OX[3], X[4], OX[4], Vin);
		fprintf(fp,"%lf %lf %lf %lf %lf %lf %lf
		%lf %lf %lf %lf %lf %lf¥n", t*dt, th1ref, X[0],
OX[0], X[1], OX[1], X[2], OX[2], X[3], OX[3], X[4], OX[4], Vin);

		t++;

// 描写
		draw_count += dt;
		if(draw_count >= 0.04){
			draw();
			reshape(width, height);
			draw_count = 0;
		}

}

void ErrorHandler(char *message, DWORD error){
		cout << message << endl;
		cout << "エラー番号 = " <<error << endl;
		ExitProcess(1);
```

```
}

void timer(int value){

        glutTimerFunc(1000*dt , timer , 0);
        move();

}

int main(){

        fp=fopen("pc-jusin.xls","w");

        srand((unsigned)time(NULL));

        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(width, height);
        glutInitWindowPosition(560, 120);
        glutCreateWindow("gl");
        init();
        glutReshapeFunc(reshape);
        glutDisplayFunc(draw);
        glutTimerFunc(100 , timer , 0);
        glutMainLoop();

        fclose(fp);

        return 0;

}

/*************************************************************************
```

```
*                                非線形システム
**************************************************************************/

void F(double X[5], double f[2]){

        f[0] = (-(g*pow(l,2)*pow(m2,2)*r*R*cos(X[1])*sin(X[1])) + (J2 + pow(l,2)*m2)*
        (r*R*X[4] - 2*(Ke*Kt + fr*R)*(X[2] - X[3])) + l*m2*r*cos(X[1])*(r*R*X[4] +
2*(Ke*Kt + fr*R)*(X[2] - X[3])) - l*m2*(J2 + pow(l,2)*m2)*r*R*sin(X[1])*pow(X[3],2))/
                ((J2 + pow(l,2)*m2)*(2*J1 + (2*m1 + m2)*pow(r,2))*R -
pow(l,2)*pow(m2,2)*pow(r,2)*R*pow(cos(X[1]),2));

        f[1] = ((2*J1 + (2*m1 + m2)*pow(r,2))*(r*R*X[4] - g*l*m2*R*sin(X[1]) + 2*(Ke*Kt +
fr*R)*(X[2] - X[3])) - l*m2*r*cos(X[1])*(-(r*R*X[4]) + 2*(Ke*Kt + fr*R)*(X[2] - X[3]) +
l*m2*r*R*sin(X[1])*pow(X[3],2)))/(R*((J2 + pow(l,2)*m2)*(2*J1 + (2*m1 + m2)*pow(r,2)) -
pow(l,2)*pow(m2,2)*pow(r,2)*pow(cos(X[1]),2)));

}

void V(double X[5], double v[2]){

        v[0] = (2*Kt*Vin*(J2 + pow(l,2)*m2 - l*m2*r*cos(X[1])))/
                ((J2 + pow(l,2)*m2)*(2*J1 + (2*m1 + m2)*pow(r,2))*R -
pow(l,2)*pow(m2,2)*pow(r,2)*R*pow(cos(X[1]),2));

        v[1] = (-2*Kt*Vin*(2*J1 + (2*m1 + m2)*pow(r,2) - l*m2*r*cos(X[1])))/
                ((J2 + pow(l,2)*m2)*(2*J1 + (2*m1 + m2)*pow(r,2))*R -
pow(l,2)*pow(m2,2)*pow(r,2)*R*pow(cos(X[1]),2));

}

void Matrix_addition(double A[5], double B[5], double C[5], double d){
```

```
	int i;

	for(i = 0; i < 5; i++){
		C[i] = A[i] + d*B[i];
	}

}

void Matrix_55(double A[5][5], double B[5], double C[5]){

	int i, j;

	C[0] = 0;	C[1] = 0;	C[2] = 0;	C[3] = 0;	C[4] = 0;

	for(i = 0; i < 5; i++){
		for(j = 0; j < 5; j++){
			C[i] += A[i][j]*B[j];
		}
	}
}

/*************************************************************************
*                          ルンゲクッタ法
*************************************************************************/

double runge_kutta(double X[5]){

	double k1[5], k2[5], k3[5], k4[5];
	double X2[5], X3[5], X4[5];
	double f1[2], v1[2], f2[2], v2[2], f3[2], v3[2], f4[2], v4[2];
```

```
F(X, f1);
V(X, v1);
k1[0] = X[2]*dt;
k1[1] = X[3]*dt;
k1[2] = (f1[0] + v1[0])*dt;
k1[3] = (f1[1] + v1[1])*dt;
k1[4] = 0;

Matrix_addition(X, k1, X2, 0.5);
F(X2, f2);
V(X2, v2);
k2[0] = (X[2] + 0.5*k1[2])*dt;
k2[1] = (X[3] + 0.5*k1[3])*dt;
k2[2] = (f2[0] + v2[0])*dt;
k2[3] = (f2[1] + v2[1])*dt;
k2[4] = 0;

Matrix_addition(X, k2, X3, 0.5);
F(X3, f3);
V(X3, v3);
k3[0] = (X[2] + 0.5*k2[2])*dt;
k3[1] = (X[3] + 0.5*k2[3])*dt;
k3[2] = (f3[0] + v3[0])*dt;
k3[3] = (f3[1] + v3[1])*dt;
k3[4] = 0;

Matrix_addition(X, k3, X4, 1);
F(X4, f4);
V(X4, v4);
k4[0] = (X[2] + k3[2])*dt;
k4[1] = (X[3] + k3[3])*dt;
k4[2] = (f4[0] + v4[0])*dt;
```

```
        k4[3] = (f4[1] + v4[1])*dt;
        k4[4] = 0;

        X[0] = X[0] + (k1[0] + 2*k2[0] + 2*k3[0] + k4[0])/6;
        X[1] = X[1] + (k1[1] + 2*k2[1] + 2*k3[1] + k4[1])/6;
        X[2] = X[2] + (k1[2] + 2*k2[2] + 2*k3[2] + k4[2])/6;
        X[3] = X[3] + (k1[3] + 2*k2[3] + 2*k3[3] + k4[3])/6;
        X[4] = X[4];

        return X[5];

}

/*************************************************************************
*                          オブザーバによる推定
*************************************************************************/

void OY(double X[5], double OX[5], double OG[5][2], double oy[5]){

        oy[0] = OG[0][0]*(X[0] - OX[0]) + OG[0][1]*(X[1] - OX[1]);
        oy[1] = OG[1][0]*(X[0] - OX[0]) + OG[1][1]*(X[1] - OX[1]);
        oy[2] = OG[2][0]*(X[0] - OX[0]) + OG[2][1]*(X[1] - OX[1]);
        oy[3] = OG[3][0]*(X[0] - OX[0]) + OG[3][1]*(X[1] - OX[1]);
        oy[4] = OG[4][0]*(X[0] - OX[0]) + OG[4][1]*(X[1] - OX[1]);

}

double observer(double X[5], double OX[5], double OG[5][2], double A[5][5], double B[5]){

        double k1[5], k2[5], k3[5], k4[5];
        double OX2[5], OX3[5], OX4[5];
        double A_OX[5], A_OX2[5], A_OX3[5], A_OX4[5];
```

```
double oy1[5], oy2[5], oy3[5], oy4[5];

Matrix_55(A, OX, A_OX);
OY(X, OX, OG, oy1);
k1[0] = (oy1[0] + A_OX[0])*dt;
k1[1] = (oy1[1] + A_OX[1])*dt;
k1[2] = (oy1[2] + A_OX[2] + B[2]*Vin)*dt;
k1[3] = (oy1[3] + A_OX[3] + B[3]*Vin)*dt;
k1[4] = (oy1[4] + A_OX[4])*dt;

Matrix_addition(OX, k1, OX2, 0.5);
Matrix_55(A, OX2, A_OX2);
OY(X, OX2, OG, oy2);
k2[0] = (oy2[0] + A_OX2[0])*dt;
k2[1] = (oy2[1] + A_OX2[1])*dt;
k2[2] = (oy2[2] + A_OX2[2] + B[2]*Vin)*dt;
k2[3] = (oy2[3] + A_OX2[3] + B[3]*Vin)*dt;
k2[4] = (oy2[4] + A_OX2[4])*dt;

Matrix_addition(OX, k2, OX3, 0.5);
Matrix_55(A, OX3, A_OX3);
OY(X, OX3, OG, oy3);
k3[0] = (oy3[0] + A_OX3[0])*dt;
k3[1] = (oy3[1] + A_OX3[1])*dt;
k3[2] = (oy3[2] + A_OX3[2] + B[2]*Vin)*dt;
k3[3] = (oy3[3] + A_OX3[3] + B[3]*Vin)*dt;
k3[4] = (oy3[4] + A_OX3[4])*dt;

Matrix_addition(OX, k3, OX4, 1);
Matrix_55(A, OX4, A_OX4);
OY(X, OX4, OG, oy4);
k4[0] = (oy4[0] + A_OX4[0])*dt;
```

```
        k4[1] = (oy4[1] + A_OX4[1])*dt;
        k4[2] = (oy4[2] + A_OX4[2] + B[2]*Vin)*dt;
        k4[3] = (oy4[3] + A_OX4[3] + B[3]*Vin)*dt;
        k4[4] = (oy4[4] + A_OX4[4])*dt;

        OX[0] = OX[0] + (k1[0] + 2*k2[0] + 2*k3[0] + k4[0])/6;
        OX[1] = OX[1] + (k1[1] + 2*k2[1] + 2*k3[1] + k4[1])/6;
        OX[2] = OX[2] + (k1[2] + 2*k2[2] + 2*k3[2] + k4[2])/6;
        OX[3] = OX[3] + (k1[3] + 2*k2[3] + 2*k3[3] + k4[3])/6;
        OX[4] = OX[4] + (k1[4] + 2*k2[4] + 2*k3[4] + k4[4])/6;

        return OX[5];

}
```