

目次

第 1 章　はじめに	3
1.1　研究の背景	3
1.2　研究の目的	3
1.3　本論文の構成	4
第 2 章　リフトについて	6
2.1　要求される仕様	6
2.2　動作原理	7
2.3　回路構造	7
2.4　使用部品	9
第 3 章　位置制御	12
3.1　位置検出	12
3.2　制御手法	12
第 4 章　位置センサレス位置制御	14
4.1　センサレスの有効性	14
4.2　センサレス制御の方法	14
第 5 章　実験	19
5.1　シミュレーション	19
5.2　検出誤差の確認	21
第 6 章　おわりに	29

参考文献	30
謝辞	31
付録 A プログラム	32

第1章

はじめに

1.1 研究の背景

当研究室では例年、高専ロボコンに出場するためのロボットの設計・製作を行っている。今年度はプラスティック製段ボールを如何に高く積み上げるかを競うための積込ロボットの製作を行った。この積み込みロボットにはリフトが搭載されており(図1.1)，このリフトが上下することでプラスティック製段ボールの箱を持ち上げたり降ろすことが出来る。リフトには上端と下端にリミットセンサが設置されており、その位置でリフトが停止するようになっている。しかし、リミットセンサによる位置検出では、センサのある場所にしかリフトを停止させることができない。任意の高さで停止させるには位置を連続的に検出できるセンサを使用した位置制御が必要になるが、そのようなセンサの使用にはコストがかかる。そこで、コストのかかるセンサを使用せずに位置制御を行う方法はないかと考えた。

1.2 研究の目的

リミットセンサによる位置検出システムは、走行中の段差などの衝撃によってプラスティック製段ボールの支持位置が変わった際、積み上げ時に繊細な高さ調整が困難であった。さらに、積み上げ時に落下させた場合積み込み幅が大きく変わってしまい積み上げの精度が悪くなる



図 1.1 積み込みロボット

恐れがあった。センサレス位置検出システムでは任意の高さでの停止が可能になるため、積み上げ時の段ボールの落下による積み込み幅の変動が無くなるのではないかと考えた。このことからロータリーエンコーダやポテンショメータといった、直接リフトの位置を検出できるセンサを使用することなく、リフトを任意の位置に停止させる制御を考案しようと考えた。

1.3 本論文の構成

1章では、本研究の背景と簡略化した概要を示す。2章では製作したりフトについて、高専ロボコンのルールをからめつつ述べる。3章では通常用いられる位置制御について述べる。4章では位置センサを使用しない制御の手法について述べる。5章ではシミュレーションソフトを

用いた実験を示す。6章で最後に本実験のまとめを述べる。

第2章

リフトについて

2.1 要求される仕様

2.1.1 ロボコンのルール

今年度のロボコンのタイトルは「ロボット・ニューフロンティア」で、新大陸の開拓をテーマとしている。今回の競技フィールドを(図2.1)に示す。新大陸では、スタート地点から運んできたプラスティック製段ボールの箱(図2.3)を高く積み上げる課題が課されている。港町と島、島と新大陸までの間の海にはロボットは接地してはならない。積み上げロボットはスタート地点と島に置いてあるプラスティック製段ボールを取りに行き高台に小さい砦を積み上げる。高台の砦が完成後、新大陸に向かい丘に砦を完成させる。これを2チームでいかに高く積み上げるかを競い合う競技となっている。

2.1.2 我々の戦略

プラスティック製段ボールの積み方を(図2.2)に示すように、同じ数の箱を積み上げる場合、箱の長手方向を縦向きにして積み上げる方が高く積み上げられる。よってリフトの可動高さを400[mm]になるよう設計、製作した。だが、長手方向を縦向きにすることで重心位置が高くなり、底面積が減るため落下時にバランスが崩れ、倒れる恐れがあるため注意する必要がある。

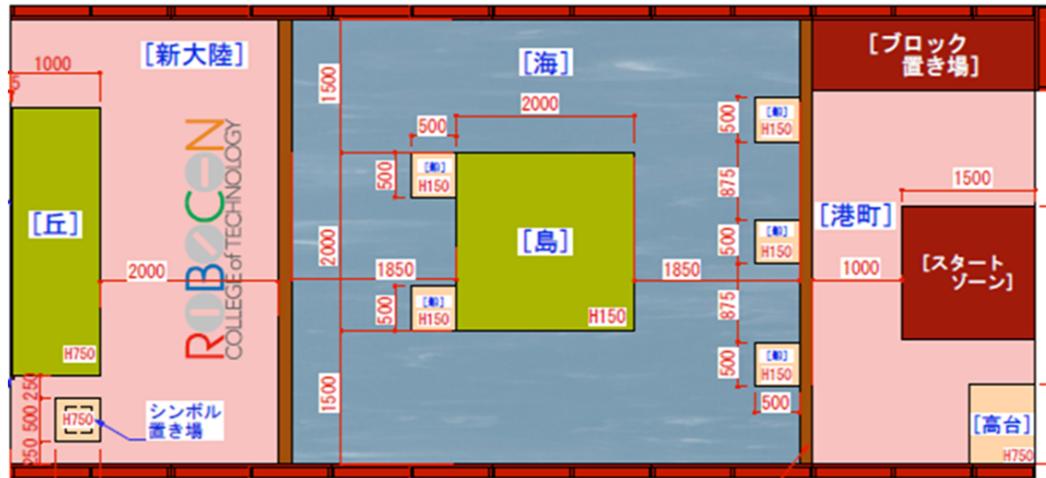


図 2.1 競技フィールド

2.2 動作原理

積込ロボットの前面にリフトが搭載されている（図 2.4）。リフト中央下部に設置されているモータによってスプロケットが回転し、チェーン駆動をする。リフト上端と下端に設置してあるリミットセンサには通過型フォトインタラプタを使用している。リミットセンサは距離が 400[mm] になるように位置が調整されているため、箱を 400[mm] の高さに持ち上げることが出来る。

2.3 回路構造

リフトに関するロボットのシステムブロック線図は図 2.5 のようになっている。arduino は図 2.6 にあるような arduinoMega を使用した。このarduino は USB ホストシールド（図 2.7）を介した Bluetooth での通信でコントローラの信号を受け取る。コントローラでの操作と上下リミットセンサの信号を受け取り、図 2.8 のモータドライバに指示を送り、リフト駆動用のモータを制御している。モータドライバは arduino から正転、逆



図 2.2 プラスティック段ボールの積み方

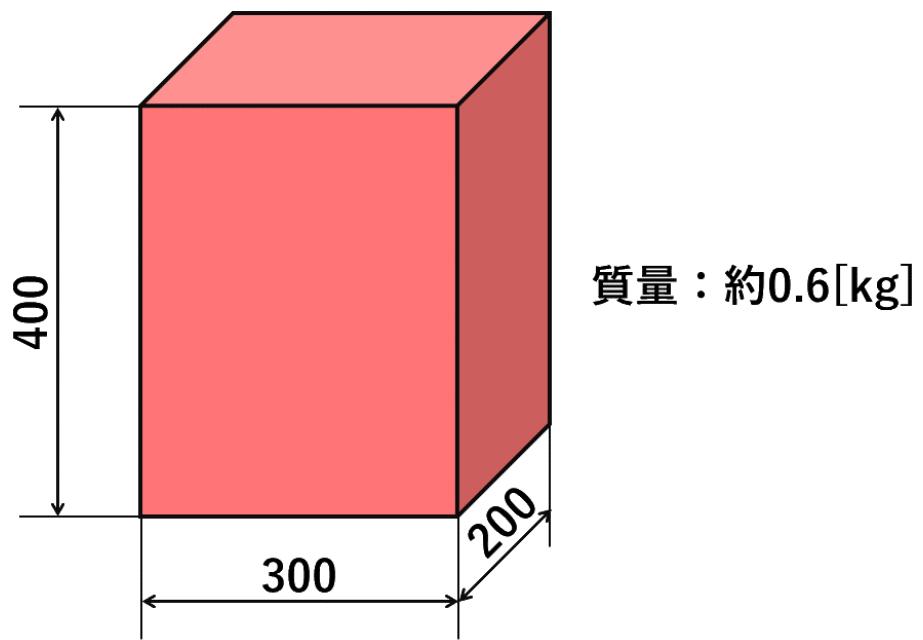


図 2.3 プラ段の箱の仕様



図 2.4 積み込みロボットのリフト

転,PWM 値を受け取り, モータに最大 12[V] の電圧を印加する.

2.4 使用部品

使用した部品の詳細を以下の表 2.1 に示す.

表 2.1 搭載部品の型番とメーカー

製品名	型番	メーカー
arduino	ArduinoMega	Arduino SRL
USB ホストシールド	BOO6J4GOOO	サインスマート
DC モータ	組み合わせモータ 323726	マクソン
コントローラ	DUALSHOCK3	ソニー
モータドライバ	-	夢考房
リミットセンサ	CNZ1023	パナソニック

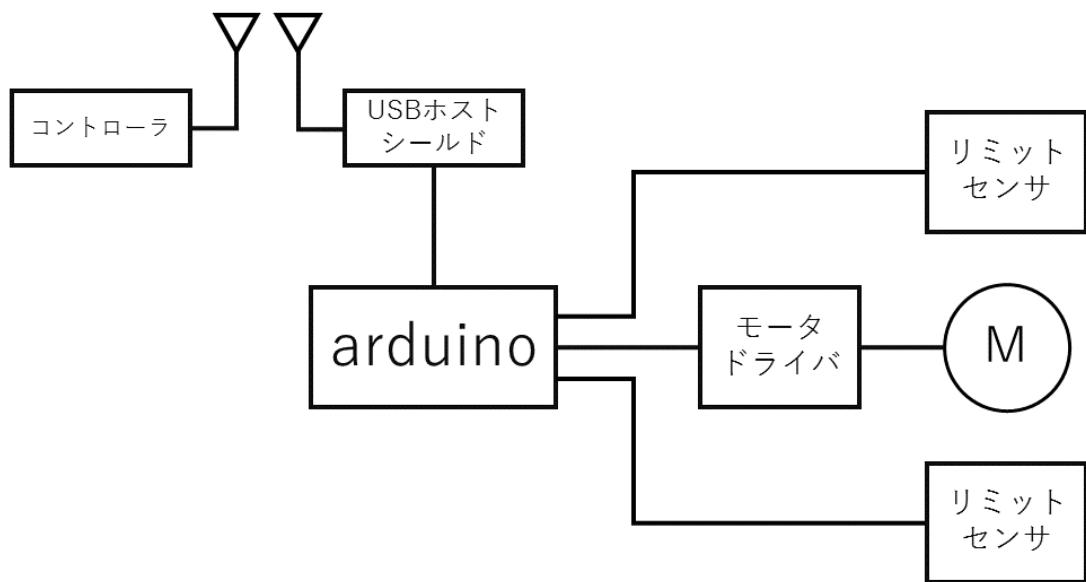


図 2.5 ロボットのシステムブロック線図

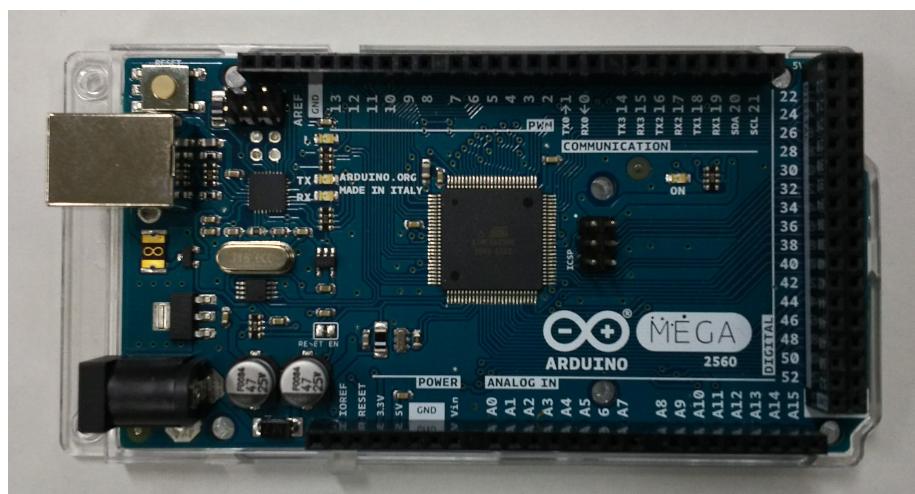


図 2.6 arduino Mega

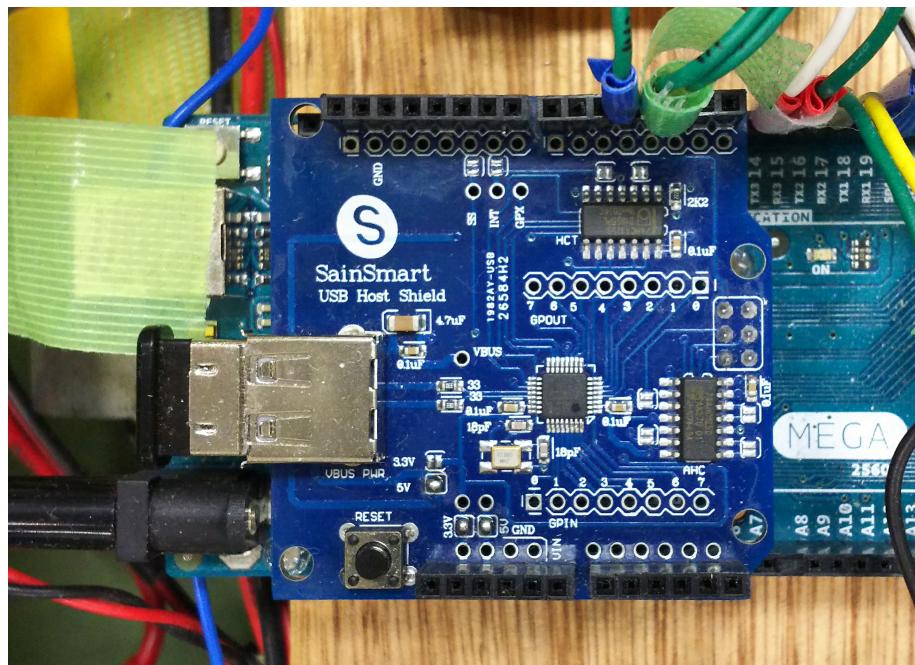


図 2.7 USB ホストシールド

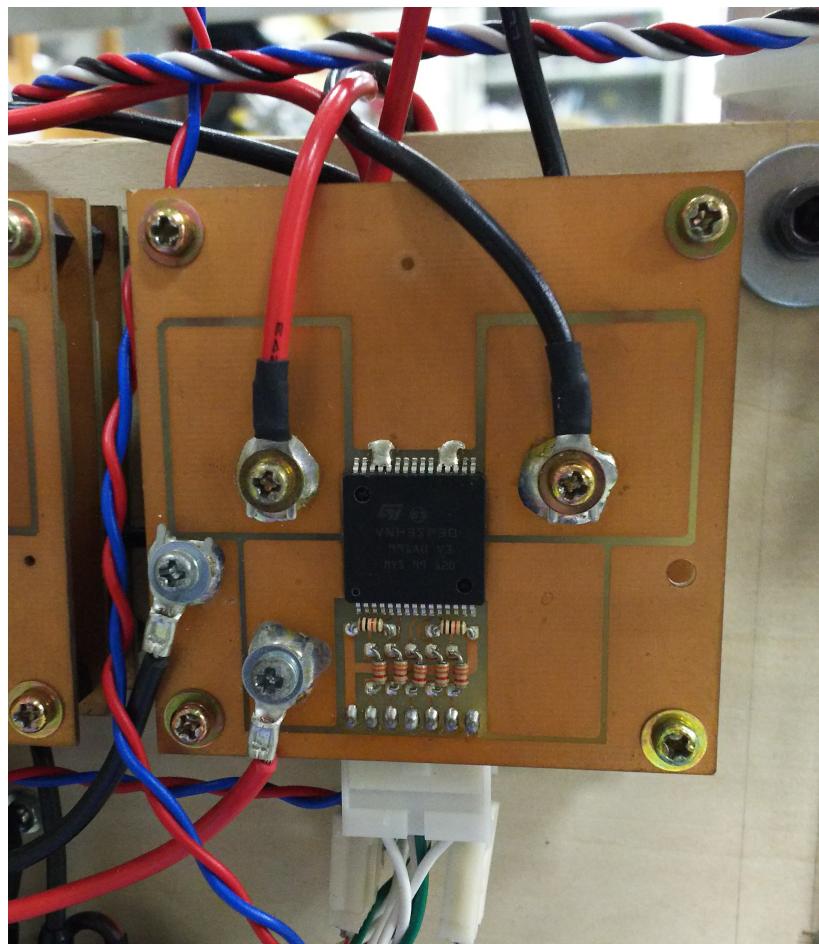


図 2.8 モータドライバ

第3章

位置制御

位置制御とは、制御対象の現在位置を何らかのセンサなどで検出し、目標の位置になるようにフィードバック制御を行うことである。

3.1 位置検出

今回のリフトのようにモータを動力とする位置制御をする際は、ロータリーエンコーダでモータの回転角度を検出し、ギヤ比やスプロケット半径から計算して制御対象の位置を特定する。このように、位置制御を行う際は位置を直接取得できるセンサを使用して、その値をフィードバックするのが一般的である。

3.2 制御手法

制御には主に以下のようないくつかの手法が使われる。

3.2.1 P I D制御

目標値と現在値の差のことを偏差といふ。偏差に比例して制御量が増える制御を比例制御(P)といい、比例定数 K_p は比例ゲインと呼ばれる。比例制御では偏差が 0 になると制御量が 0 になるため、いつまでたっても偏差が残り続ける事がある。この残留偏差をなくすため、偏差を時間で積分したものを K_i 倍して制御量に加えると、残留偏差のある時間が長いほど制御量が増えていき最終的に偏差が無くなる。これを

積分制御(I)といい, K_i は積分ゲインと呼ばれる。偏差が急激な変化をする動き始めと動き終わりには、偏差を微分した値が大きくなる。微分に比例して制御量を大きくすると機敏な動作をするようになり、早く目標値に到達する。これを微分制御(D)といい、この比例定数 K_d は微分ゲインと呼ばれる。これら P,I,D 制御を組み合わせた基本的な制御手法が PID 制御(図 3.1)である。

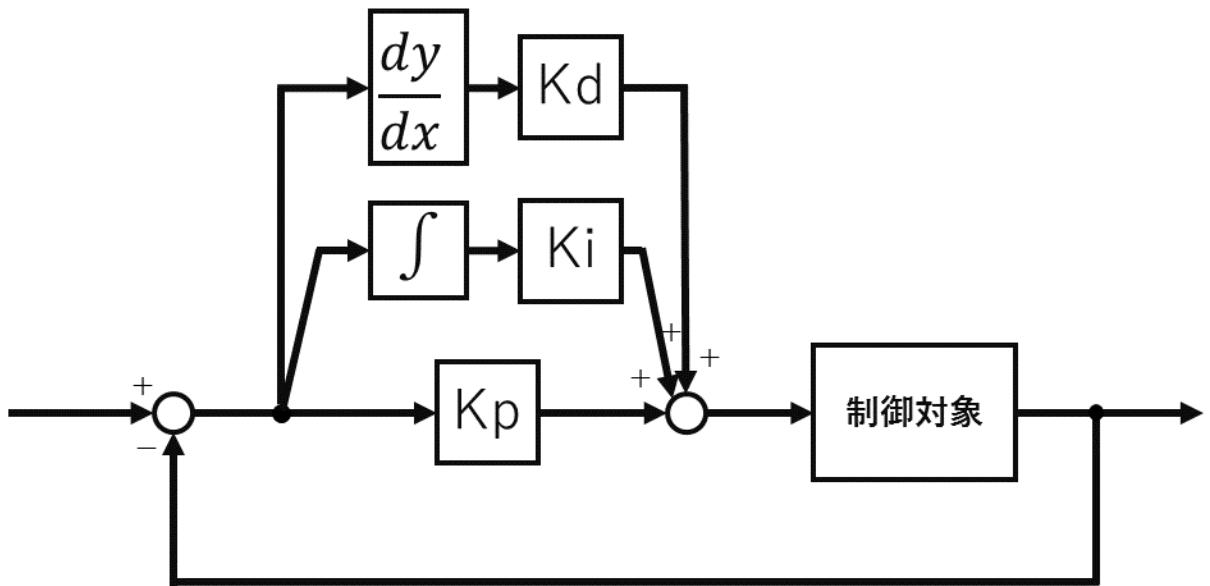


図 3.1 PID 制御のブロック線図

3.2.2 極配置法

システムは式(3.1)のような状態方程式という形で表すことが出来る。状態方程式はシステムへの入力 u , システムからの出力 y , システムの状態量 x の関係を表した方程式である。システムは伝達関数で表され、その分母は特性方程式という。特性方程式の解は極と呼ばれ、その値によって応答速度や行き過ぎ量などの特性が変化する。状態方程式で表されたシステムに対し、状態量 x に一定の値(ゲイン)を掛けて入力にフィードバックすることで極を変化させることができる。極を望ましい値に設定するフィードバックゲインを探る方法を極配置法という。

$$\begin{aligned}\dot{x} &= \mathbf{A}x + \mathbf{B}u \\ y &= \mathbf{C}x + \mathbf{D}u\end{aligned}\tag{3.1}$$

第4章

位置センサレス位置制御

4.1 センサレスの有効性

上述したように今回のリフトのような位置制御を行う場合、通常、モータの回転角度を検出するロータリーエンコーダを使用して移動量を算出する。しかし、ロータリーエンコーダを使用するにはそれ専用の取付部を新しく設計し取り付けるか、既にロータリーエンコーダと一体化しているモータを購入し使用する必要がある。ロータリーエンコーダを取り付けるにはブラケットや軸継手が必要になり、さらに取付精度も要求される。一体化されたモータなら精度は確保されているが、一般に高価である。このように、ロータリーエンコーダを後から取り付ける場合は金銭的、時間的にコストがかかる場合が多い。今回提案する位置制御はロボットに取り付ける専用のパーツを使わないとため、実現できればそれらのコストを減らすことができる。

4.2 センサレス制御の方法

モータの電流、電圧、回転速度には相関があり、二つの値が定まると残りの一つの値も定まる。電圧の値は操作量なので既知であるため、電流の値が取得できれば、モータの回転速度を求め、それを積分して回転角度を求め、そこから移動量を算出することが出来る。必要なものは電流センサを使った電流計測回路のみなので、ロータリーエンコーダを使

うよりも後付けを簡単に行うことができる。

4.2.1 制御対象のモデル

リフトの簡略モデルを図4.1に示す。リフトの運動方程式は式(4.2)のようになる。

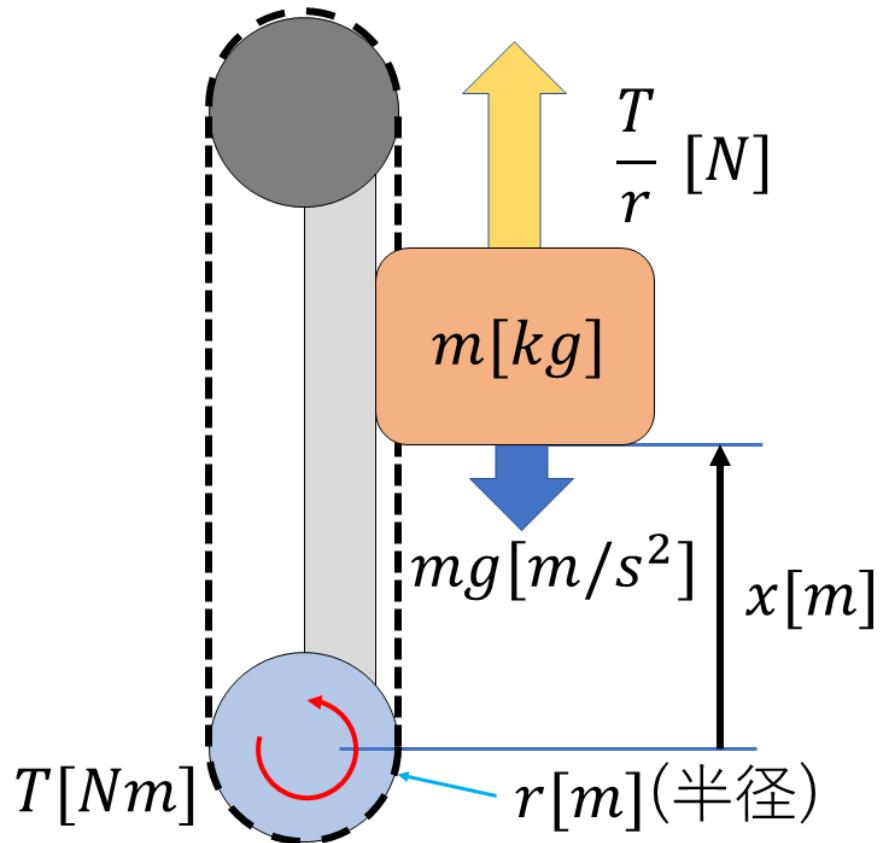


図4.1 リフトの簡略モデル

$$mv = \frac{T}{r} - mg \quad (4.1)$$

$$(4.2)$$

式(4.2)をトルクについて変形すると、式(4.4)のようになる。

$$T = rm(\dot{v} + g) \quad (4.3)$$

$$(4.4)$$

モータ回路の簡略モデルを図4.2に示す。モータの運動方程式は式(4.6)のようになる。

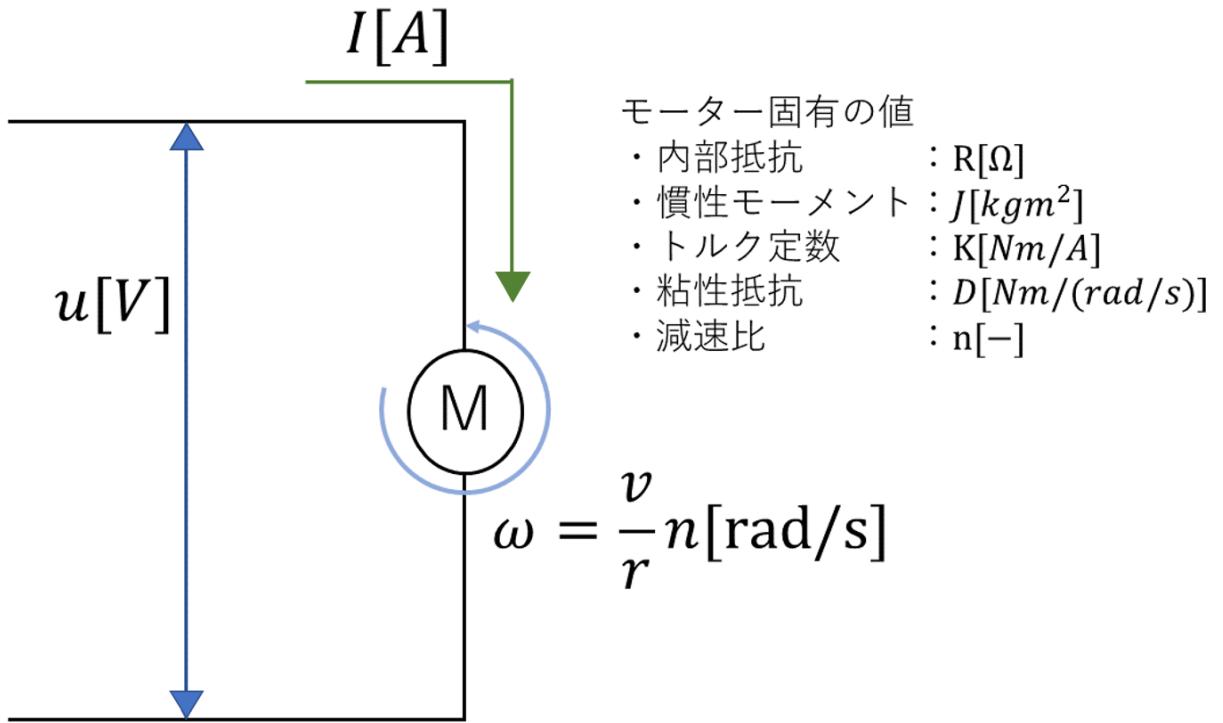


図4.2 モータ回路簡略モデル

$$RI + K\omega = c \quad (4.5)$$

$$J\dot{\omega} + \frac{D}{n}\omega - \frac{T}{n} = KI \quad (4.6)$$

モータの回転数と歯車比、スプロケット半径との関係は式(4.7)で表される。

$$\omega = \frac{v}{r} n \quad (4.7)$$

リフトの上昇速度を状態量に、電流値を観測方程式とする状態方程式を立てる。式(4.2), 式(4.6)より、リフトの状態方程式は式(4.9)のようになる。

$$\dot{v} = -\frac{RD + K^2}{RJ + r^2m/n^2}v + \frac{KRn}{RJn^2 + r^2m}u - \frac{g}{Jn^2/r^2m + 1} \quad (4.8)$$

$$I = -\frac{Kn}{Rr}v + \frac{1}{R}u \quad (4.9)$$

リフトの状態方程式の観測方程式を変形し、リフトの速度を電流と電圧で表すと式(4.10)のようになる。

$$v = -\frac{Rr}{Kn}I - Ru \quad (4.10)$$

式(4.10)には質量 m が含まれていない。よって速度推定はリフトの荷重によって影響されないと考えられる。

4.2.2 制御系設計

ブロック線図を図4.3に示す。

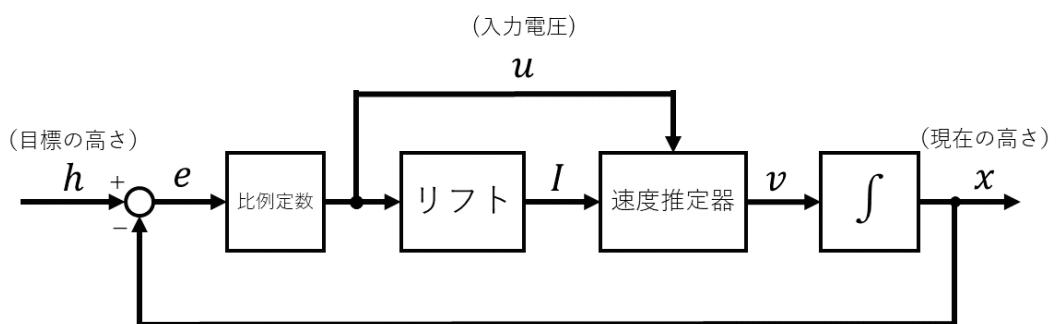


図4.3 ブロック線図

リフトには、目標値と現在値の偏差に比例した値を入力する比例制御を行っている。通常の位置制御であればリフトの位置を直接検出してフィードバック制御を行う。しかし今回のセンサレス制御では位置を検出できない。そのため、電流センサから得た電流と入力の電圧から速

度測定器でリフトの移動速度を求め、それを積分した値を現在値としてフィードバックする。速度推定器は式(4.11)で表される。

$$v = -\frac{Rr}{Kn}i + \frac{R^2 r}{Kn}u \quad (4.11)$$

速度測定器で算出した移動速度は実際のリフトの移動速度とは異なる可能性がある。

第5章

実験

5.1 シミュレーション

シミュレーションを利用して、設計した制御系の性能を確認する。

シミュレーションソフト

シミュレーションには数値計算ソフト「Scilab」に付属しているビジュアルモデリングソフト「Xcos」を使用する。Xcosで組み立てたブロック線図を図5.1に示す。この際、リフトにかかる電圧の絶対値の最大は実際と同じく12[v]に制限した。

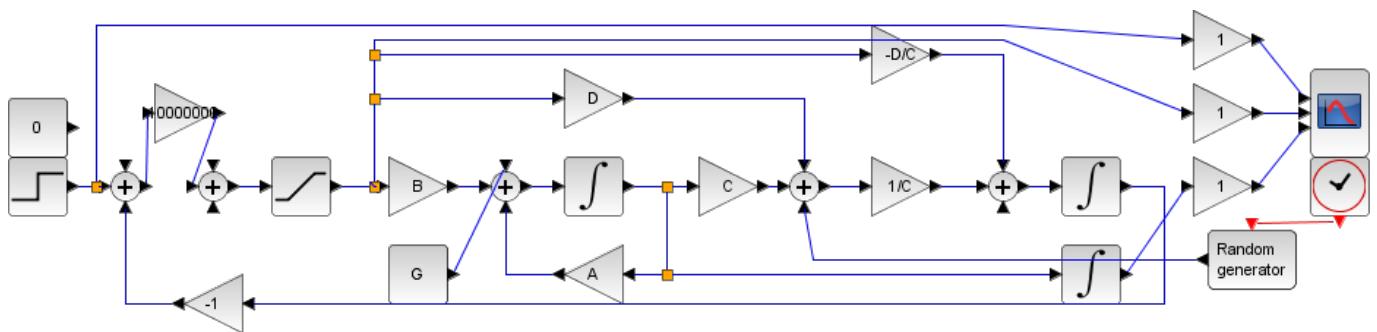


図5.1 Xcosで組み立てたブロック線図

シミュレーション結果

シミュレーションの結果を図5.2に示す。

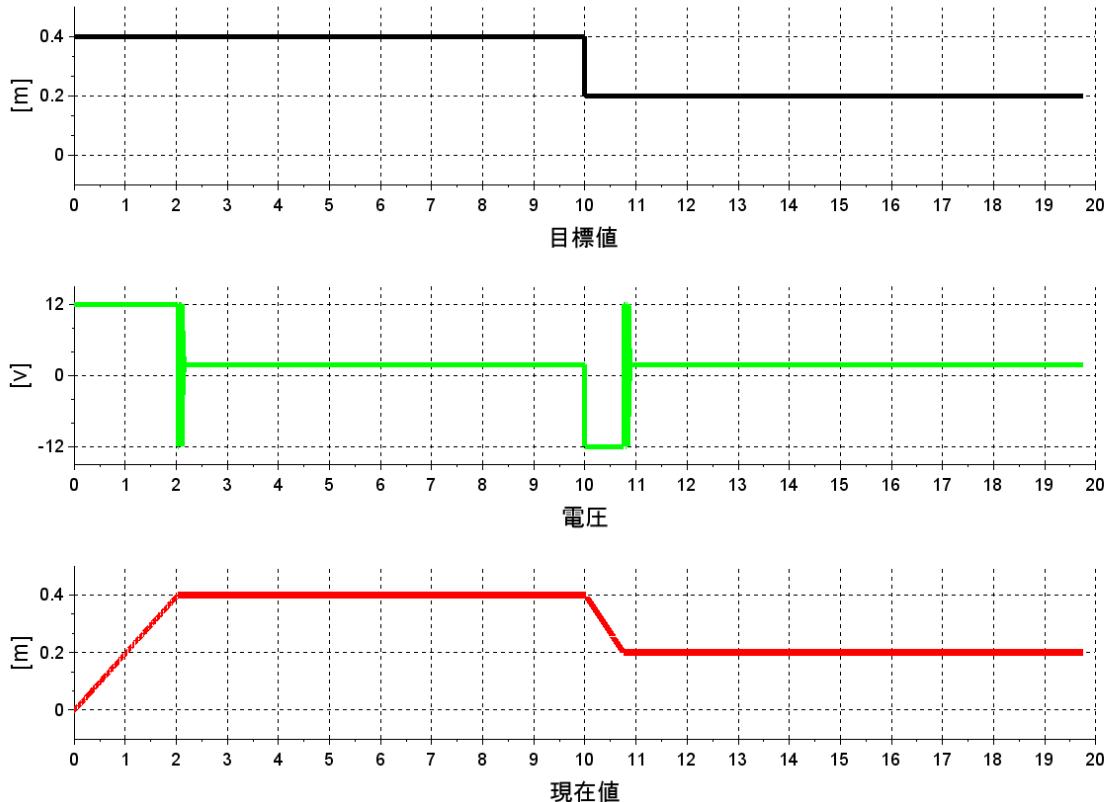


図 5.2 入力した目標値による電圧と位置の時間変化

荷重の影響

リフトは箱を持つ個数により荷重が変化する、荷重の影響を見るためリフトの質量の値を2倍にした。その際のシミュレーション結果を図5.3に示す。

荷重によって電圧は2倍になったが、現在値の最終値に変化はなかった。

測定誤差の影響

実際に電流を測定した時にはノイズがあると考え、ノイズの影響を見るため電流の値に1[A]の誤差を加えてシミュレーションした。その結果を図5.4に示す。

電流の値に誤差があると、実際のリフトの速度と推定する速度にずれが生じることが分かった。

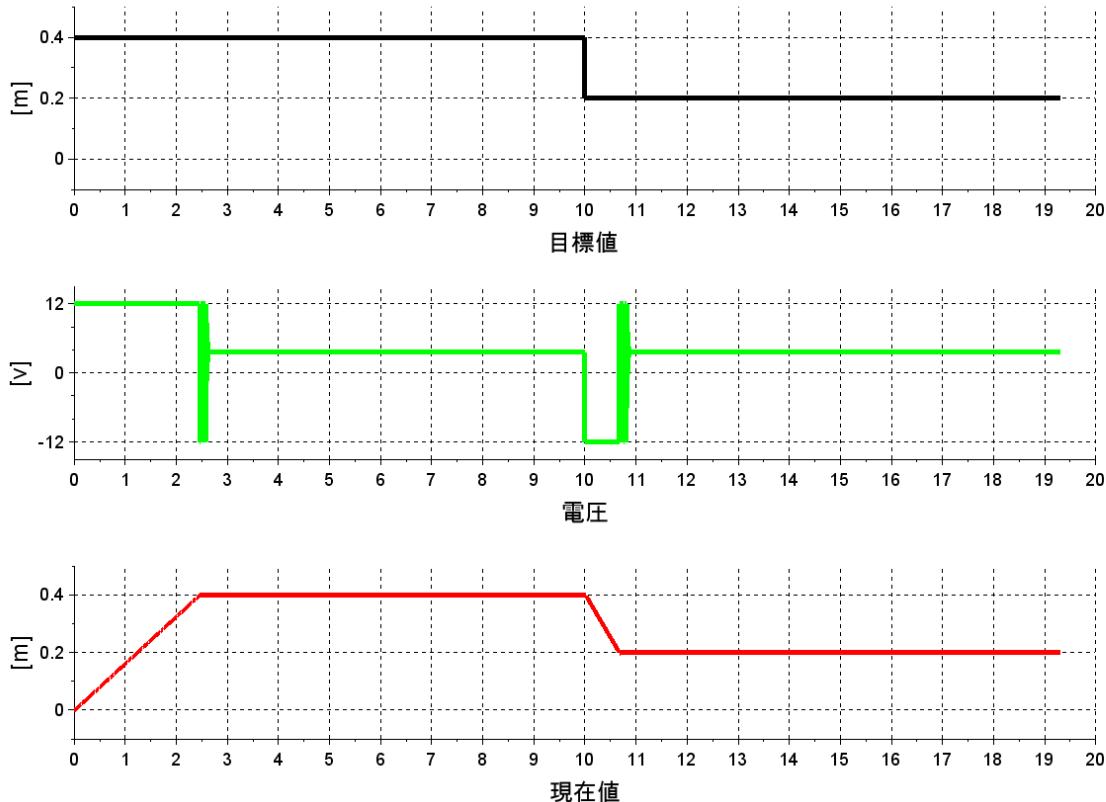


図 5.3 荷重による影響

測定ノイズの影響

実際に電流を測定した時にはノイズがあると考え、ノイズの影響を見るため電流の値を分散が $1[A]$ の正規分布になるようにした。その際のシミュレーション結果を図 5.5 に示す。

電圧のグラフが激しく振動している為、塗りつぶされて見える。これは電流のノイズの影響だと考えられるが、ノイズの平均値が 0 ならば現在位置に影響はなかった。

5.2 検出誤差の確認

電流センサが検出する電流と実際に流れている電流に誤差があると、リフトの速度推定にも誤差が生じてしまい、時間が経つにつれリフトの位置がずれていってしまう。ここでは電流センサで検出した値がどれだけの測定誤差を持つのか確認する。

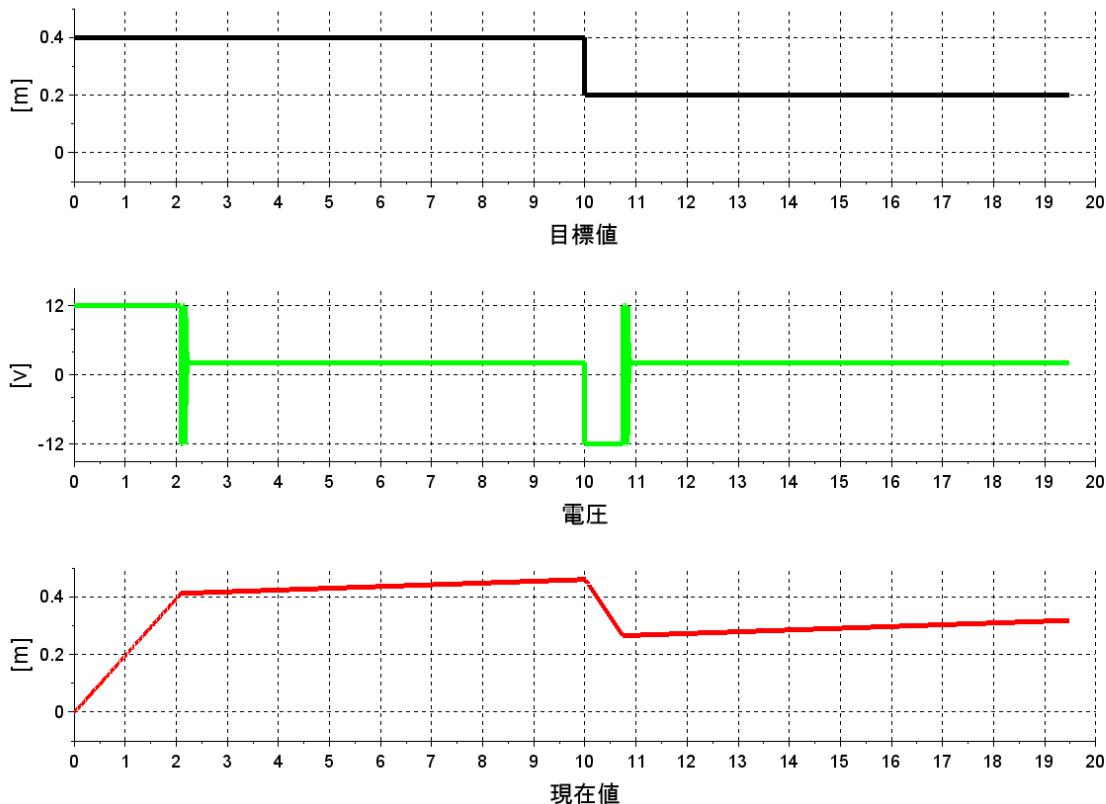


図 5.4 測定誤差の影響

使用機器

電流センサを内蔵したモータドライバを使用した。使用したモータドライバを図 5.6 に示す。使用した機器一覧を表 5.1 に示す。

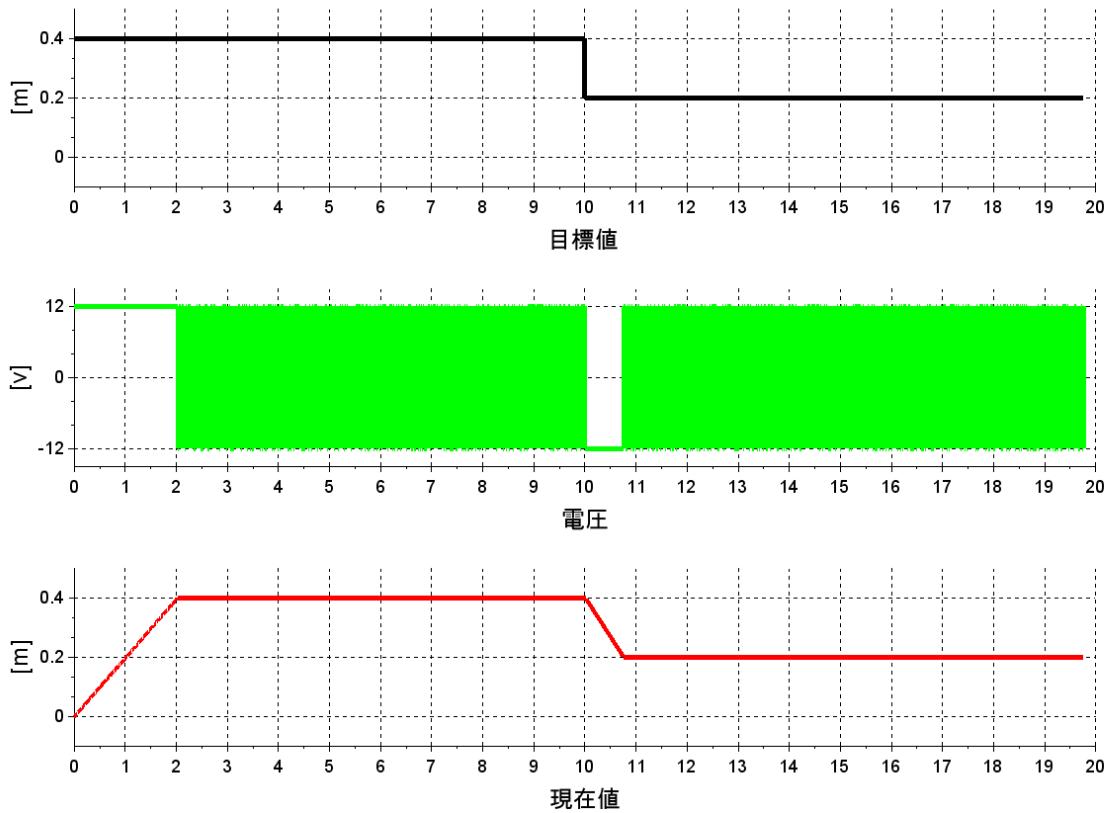


図 5.5 測定ノイズの影響

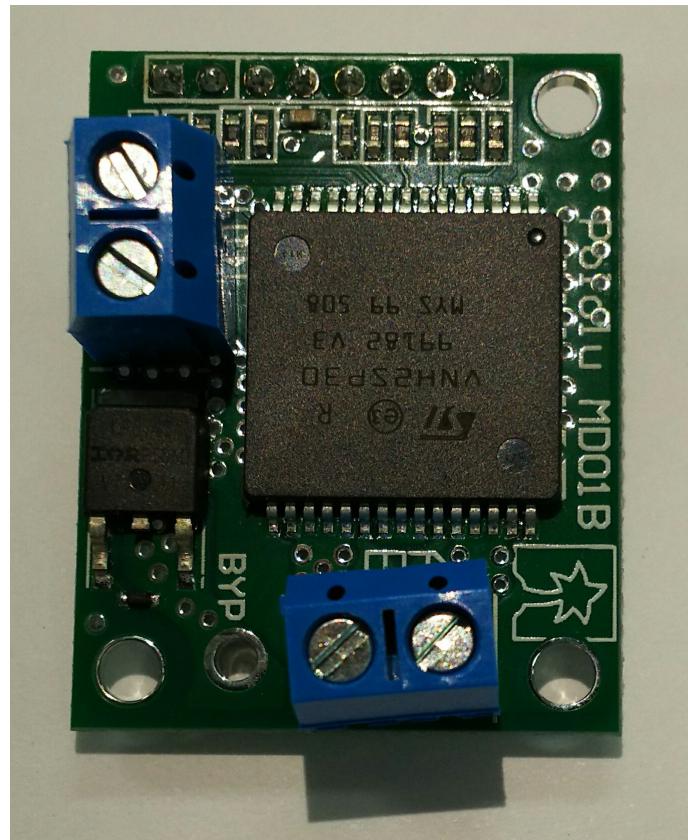


図 5.6 電流センサ内臓モータドライバ

表 5.1 搭載部品の型番とメーカ

製品名	型番	メーカ
arduino	ArduinoUno	Arduino SRL
DC モータ	組み合わせモータ 323726	マクソン
電流センサ内臓モータドライバ	MD01B	Pololu
テスター	VOAC86	IWATU

計測方法

- 1). arduino で, モータに常に電圧を 12[v] 印加するように, 電流センサの値を 1 ミリ秒ごとに取得するようにプログラミングする.
- 2). 10 秒以上データを記録したら, 電流の平均値と分散を算出する. 突入電流といい, モータに電圧を印加した直後は大きな電流が瞬間に流れれるため, しばらくたった定常状態の値を計算に使用する.
- 3). テスターを用いて実際の電流を計測する.
- 4). 電流センサでの平均値とテスターでの計測値の差を求め, ノイズの平均値とする.

電流計測

電流を計測した結果を図 5.7 に示す.

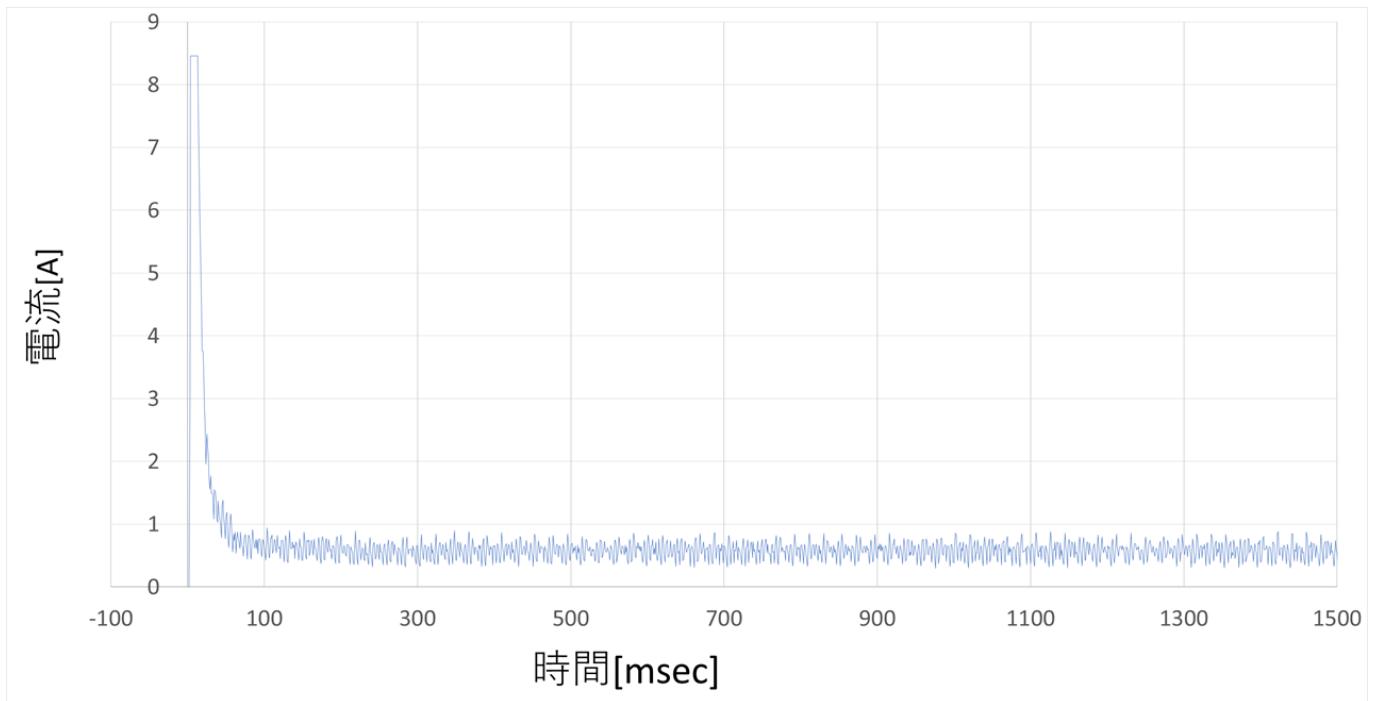


図 5.7 電流の時間変化

検出した電流の平均は $0.575[A]$, ノイズの分散は $0.016[A]$, テスターでの計測結果は $0.232[A]$ となつた. よってノイズの平均は $0.343[A]$ となつた. モータに印加する電圧を負にして, 逆転させた場合の計測結果を図 5.8 に示す.

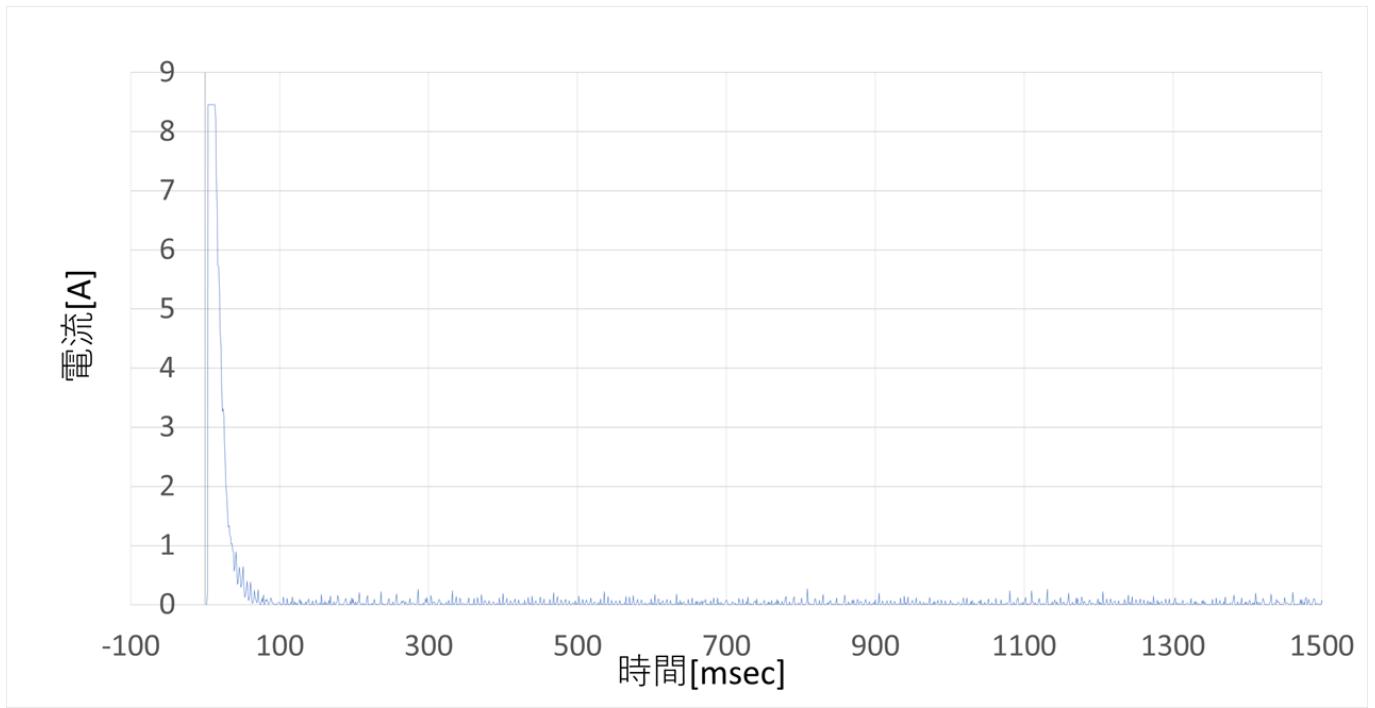


図 5.8 電流の時間変化

検出した電流の平均は $0.027[A]$, ノイズの分散は $0.002[A]$ となつた. よつてノイズの平均は $-0.205[A]$ となつた. 正転時と逆転時に電圧の違いが表れたが, arduino の AD 変換器に原因がある可能性を考え, オシロスコープで電流センサから出力される電圧を直接観察した. その結果を図 5.9, 図 5.10, 図 5.11 に示す.

オシロスコープでの観察でも電圧の値に差がみられた.

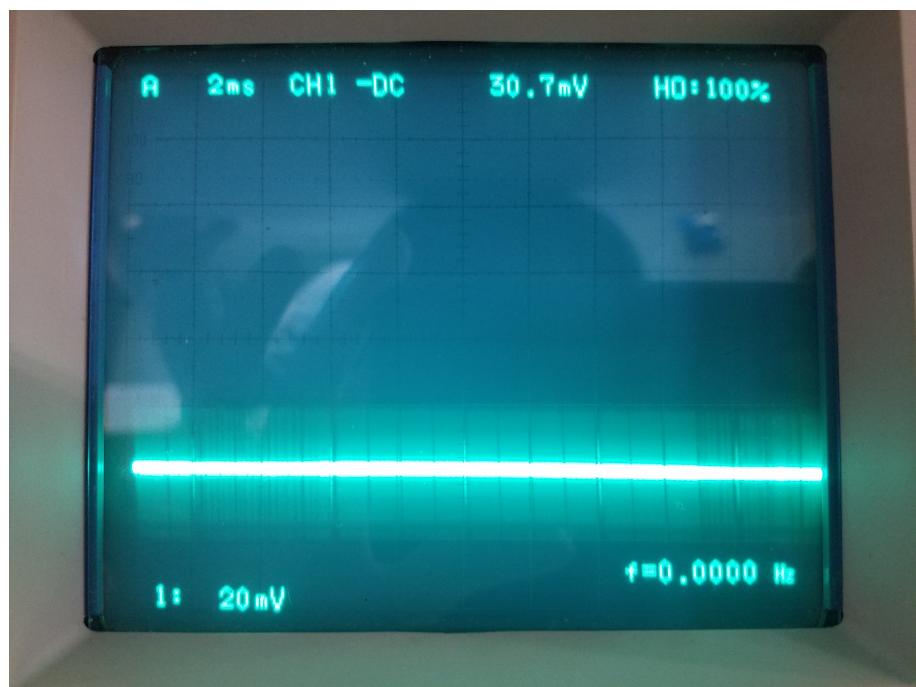


図 5.9 無回転時の電圧

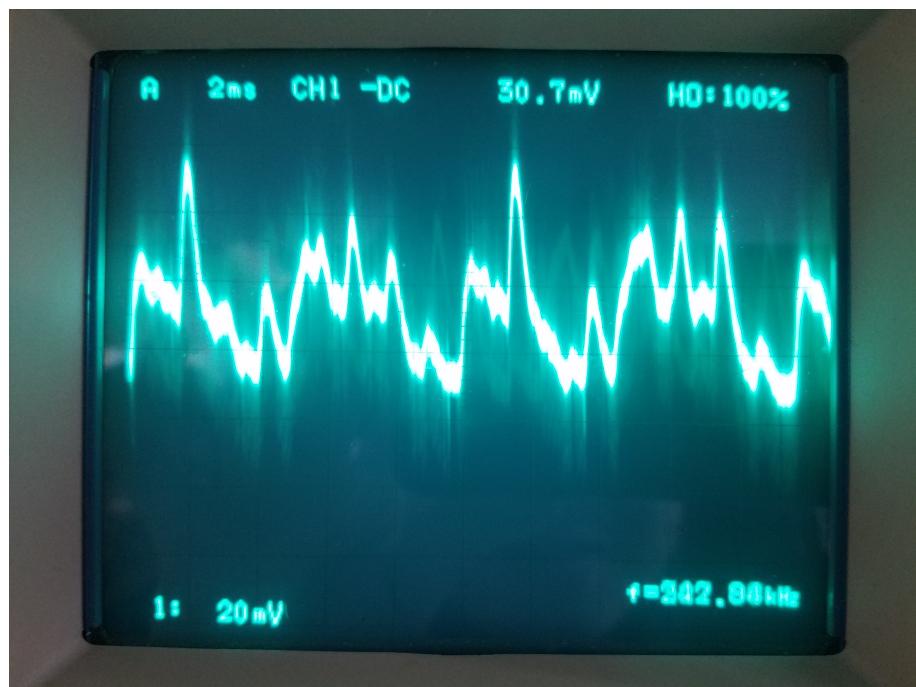


図 5.10 正転時の電圧



図 5.11 逆転時の電圧

第6章

おわりに

シミュレーションを使用して、センサレス制御は可能であることを確認することができた。実験に使用したモータドライバには特性があり、正転と逆転時に異なる電流値を出力することが分かったため、この制御を実際のリフトに適用する際には、電流センサのノイズによって引き起こされる速度推定誤差とモータのモデル化誤差を何らかの方法で補正して解決する必要があると考えられる。

参考文献

- [1] 熊谷正朗, 状態フィードバックとオブザーバ, www.mech.tohoku-gakuin.ac.jp/rde/contents/course/controlII/statefeedback.html, 2011/02/10
- [2] 明石一・今井弘之, 制御工学演習, 共立出版, 2014/04/15

謝辞

本論文作成にあたり研究の考え方, 方法のまとめ方など長期にわたつて熱意のあるご指導, ご鞭撻していただいた, 伊藤恒平教授に厚く御礼申し上げます。

特に制御においても論文の書き方においても論文を何度も読んでいただき, 指導していただいた伊藤恒平教授, 伊勢大成講師に大変ご苦労をかけてしましましたことにも心よりお詫び申し上げます。

その他, 助けていただいた多くの皆様に心から感謝しております。ありがとうございました。

付録 A

プログラム

saucecode A.1 電流計測プログラム (Arduino Uno)

```
1 //電流計測
2 \#include<FlexiTimer2.h>
3
4 unsigned long timer_a = 0;//マイコン起動時からの時間を記録する変数（ミリ秒）
5 const int ts = 1;//サンプリング時間（ミリ秒）
6
7 const int pinI = 1;//アナログ1番ピンを使用する
8 int current = 0;
9
10 int pwm = 0;//この値を0～255まで変更することで電圧を0～1.2（v）まで変化させることが出来る
11
12 void setup() {
13
14     //シリアル通信の速度を設定する シリアルモニターの数値と合わせること
15     Serial.begin(115200);
16
17     //アナログ入力の基準電圧を1.1（v）に変更する
18     analogReference(INTERNAL);
19
20     //タイマー割り込みの設定
21     FlexiTimer2::set(ts, motorCalculat);
22
23     //タイマー割り込みの開始
24     FlexiTimer2::start();
25
26     //ピンを出力に設定する
27     //5番ピンはPWM出力に対応している"""
28     pinMode(2, OUTPUT);
29     pinMode(3, OUTPUT);
30     pinMode(5, OUTPUT);
31 }
32
33 void loop() {
34
35     //アナログピンから電流を読み込む
36     current = analogRead(pinI);
37 }
```

```
38 //ピンから信号を出力する
39 //2と3を入れ替えると逆転する
40 digitalWrite(2, HIGH);
41 digitalWrite(3, LOW);
42 analogWrite(5, pwm);
43 }
44
45 //1ミリ秒毎にタイマー割り込みで呼び出される関数
46 void motorCalculat() {
47     Serial.print(pwm);
48     Serial.print(' ');
49     Serial.println(current);
50     timer_a += ts;
51 }
```
