

卒業論文

力制御による電動スライダの制御

1P — 51ページ 完

平成13年2月28日提出

指導教官 長尾 高明 教授

高知工科大学

知能機械システム工学科

1010145 現田 雅也

目次

1. 序論	5
1.1 研究の背景と目的	6
1.2 従来の研究	7
2. 加工の臨場感通信システム	8
2.1 臨場感通信システムとは	9
2.2 臨場感通信システムの必要性	10
3. 操作システムの構成	11
3.1 電動スライダ	12
3.2 ドライバ	14
3.3 力センサ	16
3.4 検定用コンピュータ	18
3.5 モータコントロールボード	19
3.6 A/D変換ボード	21
3.7 ストレイン・アンプ	23
4. 実験	25
実験全体概要図	26
4.1 電動スライダ基本制御実験	27
4.1.1 目的	
4.1.2 結果と考察	
力センサの検定実験	29
4.2 力による電動スライダの制御実験	31
4.2.1 目的	
4.2.2 結果と考察	

目次

5. 今後の展望	32
謝辞	34
付録	35

第 1 章

序論

1.1 研究の背景と目的

最近の急激なハード、ソフトの発展によって、ネットワークの重要性が改めて認識されている。発展したネットワークは人と人との物理的な距離を越えて結びつける第2の道というべき存在になりつつある。加工においても実際の加工の現場と操作者を切り離しネットワークによって2つの場所を結びつける技術が注目されてきている。

加工の現場と操作室を別にすることは、操作の環境を快適なものにできるという利点があるが、利点はそれだけでない。例えば設計者はデザインルームにいながらにして実験的な加工を行うことができるようになる。また、ネットワークによる情報の標準化は操作者に対してコンピュータの手助けを受け易くするだろう。つまり、これまで機械に近すぎた人にはもっと離れるチャンスを、機械に遠すぎた人にはもっと近づくチャンスを与えてくれる技術といえるだろう。

また、技術の進歩は距離的に遠いだけでなく、通常の生活の感覚とはまったく別の物理法則が支配する世界へも足を踏み入れている。電子顕微鏡や STM (Scanning Tunnel Microscope: 走査型トンネル顕微鏡) などの開発によって、マイクロあるいは原子レベルの世界を「見る」ことができるようになってきた。さらに最近では、マイクロ・マシンあるいはナノ・ロボットによってこのようなマイクロの世界で物に「さわる」、あるいは、「加工する」ことができるようになってきている。いわゆる「マイクロマシニング」という研究である。マイクロマシニングを行う際には、実際の音を聞くことはできない。もちろん人間の触覚も役に立たない。このような環境で加工を行う時には何をもって「音」とし、何をもって「触覚」とすればよいのであろうか。

上で述べたように操作室と加工現場が物理的に異なる場所にある場合において、あたかもその場にいるかのように加工ができるようなシステムが必要とされている。この環境を実現するシステムが加工の臨場感操作システムである。

操作者の操作ミスをいかに防ぎ、使い易いシステムを作るかが課題となってきた。3次元座標系で直接操作を与え、かつ加工状態を手の感覚で感じることができる操作システムが必要ではないかと考えられる。

1.2 従来の研究

3次元の位置と送り、力を受けるという構造はロボットに関してのマスター・スレーブモデル、特にバイラテラル制御と呼ばれる技術が参考となると思われる。

第2章

加工の臨場感通信システム

2.1 臨場感通信システムとは

加工の臨場感通信システムとは、工作機械を操作する操作者と、工作機械および被加工物が異なる世界にあるときに、操作者があたかも目の前で加工をしているように感じさせるシステムである。このシステムを実現するには実際の加工の際に感じているように我々に感じさせる必要がある。

さて、本論文において「加工における異なる世界」とは、人間が備える感覚器官では直接知覚することができない遠隔地や微小世界、あるいは、真空中、宇宙空間、水中等の環境を意味する。これらの世界と人間の存在する空間とを加工反力、加工熱、加工音、加工状態(視覚)等の情報と機械の制御情報とによって、有機的に結合するのが本システムである。このような環境に設置された対象を加工、操作しようとするとき次のような問題が生じる。

1. 対象が遠隔地に存在する場合、通信が高速であっても無視できない遅れ時間を生じる。
2. 対象が真空中に存在する場合には、工作物から加工音が伝わってこない。
3. 微小世界では支配する物理法則が異なる。具体的には、慣性力よりも摩擦や粘性力、静電気力の影響が大きくなり、人間が通常存在する世界の感覚では微小世界の対象を自由に操作できない。

これらの問題を解決するシステムが加工の臨場感通信システムである。このシステムを用いることで以下のような利点が生じる。

1. 対象となる被加工物がどこにあっても、同じ感覚で操作することができる。
2. 対象となる被加工物がどのような大きさのものであっても、同じ感覚で操作することができる。
3. 対象となる被加工物がどのような大きさで、どこにあっても、同じ操作室で作業することができる。

以上の利点がある。

2.2 臨場感通信システムの必要性

加工の臨場感通信システムにおける欠点を解決するためには、さらになにかの情報を操作者に伝えなければならない。これに関して、人間が実際に工具を使って加工することについて考えてみたいと思う。加工とは根本的に考えるならば、「被加工物に工具を接し、力を与え、かつ工具の位置を動かすことによって被加工物の形状を好ましい形状に変化させること」といえるだろう。この時工具と被加工物を持っているのは手である。手は3次元の位置がわかり、また力覚、触覚、温覚といった感覚を備えている。したがって、手を媒介することによって、位置を直観的に知ることができることはもとより、加工反力によってもたらされる情報は非常に多くの情報を操作者に与えてくれるだろう。例えば手に返ってくる振動によって加工状態がわかる。また、加工の際の力加減といったものも感覚的に自由に変えることができる。

以上のようなことから、操作者の手からの力情報を受け取り、操作者に工具の位置、反力をフィードバックするような装置を導入することによって、加工の臨場感通信システムの使い勝手を改善することができるのではないかと思われる。

第3章

操作システムの構成

3.1 電動スライダ

リニアガイド、研磨ボールねじ、ステッピングモータを一体化したオレンタルモーター社製の SPF60B10-3SD を使用した。この電動スライダは、工作機械が直行座標系であるのでそれに合わせて直線運動のできる機構として用いた。また両端には衝突防止のためにリミットセンサを取り付けた。電動スライダの仕様を表3.1に示す。

表3.1 電動スライダ仕様

ストローク	(mm)	300
最大速度	(mm/s)	500
水平方向最大可搬質量	(kg)	10
最大負荷モーメント	N・m(kgfm)	Mp:35 My:23 Mr:9
最大推力	N(kgf)	80
最大励磁保持力	N(kgf)	80
繰り返し位置決め精度	(mm)	±0.01
分解能	(mm/step)	0.01
ロストモーション	(mm)	0.03以下
質量	(kg)	4.2
使用周囲温度		0℃～+40℃
センサ		HOME, +LS, -LS 3個添付(可動)

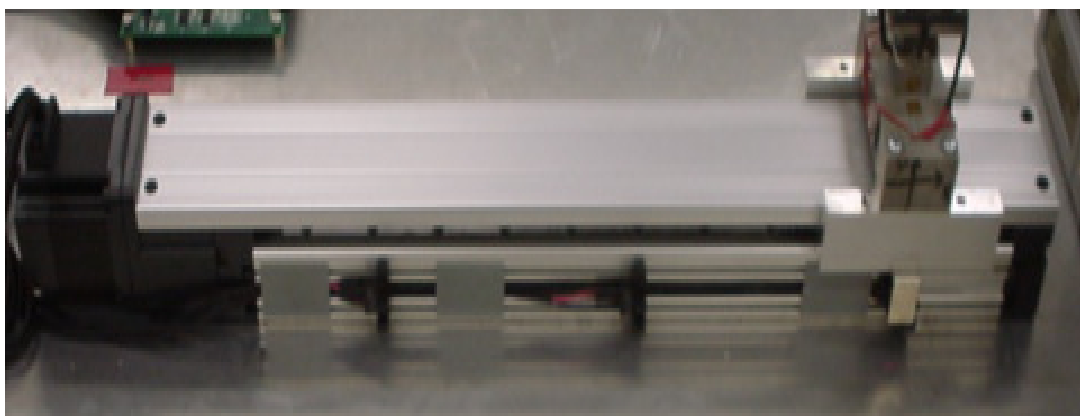


図3. 1 電動スライダ

3.2 ドライバ

モータを制御するためのドライバとして同じくオリエンタルモーター社製のものを使用した。

表3.2 ドライバ仕様

電源入力	電圧	100V～115V相当
	周波数	50/60Hz
	電流	5A
駆動方式		正弦波 PWM方式
最大入力パルス周波数		250kHz
入力形式	等価入力抵抗	220 Ω
(フォトカプラ)	入力電流	7～20mA
	信号電圧	H:4～5V, L:0～0.5V
出力形式	外部使用条件	DC30V以下、15mA以下
(フォトカプラ・オープンコレクタ)	ドライバ出力	HD26C31相当
冷却方式		自然空冷方式
使用周囲温度		0℃～+50℃
質量		0.8kg



図3. 2 ドライバ

3.3 カセンサ

操作者の手からの力を検出するために、3軸力を検出するためにのセンサを用いる。3次元ジョイスティック自体がストローク300mm程度であることや、人の手が使うものであることを考えると、なるべく小さなサイズにせざるをえない。このセンサは研究室のワイヤ・カットを用いて製作した。カセンサの仕様を表3.3に示す。

表3.3 カセンサの仕様

材質	アルミ
構造	平行平板構造
定格	2 [kgf]
出力	X軸 170[μ st.]
	Y軸 337[μ st.]
	X軸 116[μ st.]

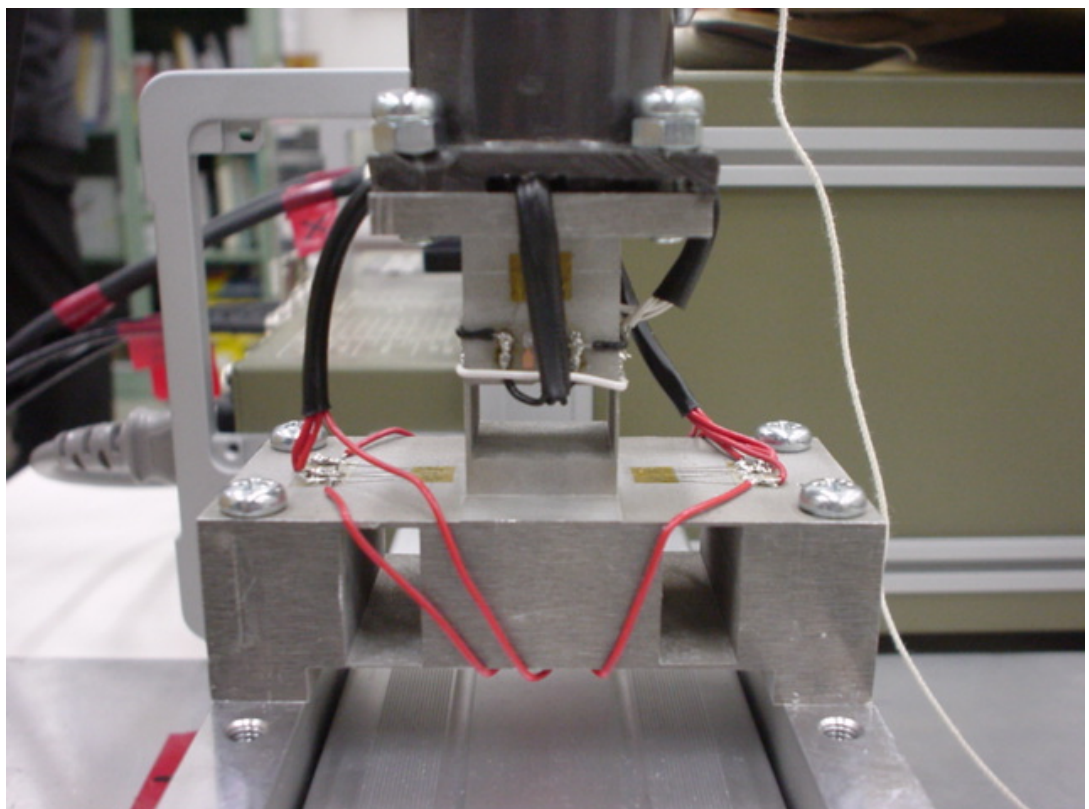


図3. 3 力センサ

3. 4 操作用コンピュータ

モータコントロールボード(PCPG-46)、A/D変換ボード(PCI-3155)を搭載した操作用コンピュータ。このコンピュータより電動スライダの制御を行う。



図3. 4 操作用コンピュータ

3.5 モータコントロールボード(PCPG-46)

最高出力周波数4.096MPPS 及び速度オーバーライド、移動量オーバーライド、直線加減速、自動S字(放物線)加減速を実現した、PCIバス対応の高機能4軸モータコントロールボード。表3.4にモータコントロールボード(PCPG-46)の仕様をします。

表3.4 モータコントロールボード(PCPG-46)仕様

専有アドレス	64アドレス使用
外形寸法	106.68×174.63mm(コネクタ部、パネル部含まず)
入力電源	DC+5V±0.25V
消費電流	Min 200mA Max 1000mA
制御機能	PRESET PULSE DRIVE (指定パルス数ドライブ) CONTINUOUS DRIVE (連続ドライブ) SIGNAL SEARCH-1, 2 DRIVE (信号検出ドライブ) 最高出力周波数…4.096MPPS 速度オーバーライド機能 移動量オーバーライド機能 直線加減速機能 自動S字加減速機能 出力パルス数管理用28Bit Counter内蔵(各軸毎) フィードバックパルス数管理用28Bit Counter内蔵(各軸毎) 偏差量自動算出機能 急停止機能(入力信号、コマンド書き込み 何れも可能) 減速停止機能(コマンド書き込み) リミット停止機能(急停止、減速停止 何れも可能) 最大4軸制御(同時ドライブ可能)
出力信号	パルス出力等…ラインドライバによる差動出力 ドライバ制御出力…フォトカプラによるオープンコレクタ出力
入力信号	機械系入力及び汎用入力…+12～24V フォトカプラ入力 ドライバステータス入力…+12～24V フォトカプラ入力 フィードバックパルス入力…ラインレシーバーによる差動入力
使用可能ドライバ	ステッピング及びサーボモータドライバ パルス列入力型(1パルス/2パルス 何れも可能) 差動入力、フォトカプラ入力、TTL入力型



図3.5 モータコントロールボード(PCPG-46)

3.6 A/D 変換ボード

ストレインアンプを通し、電圧に変換された力センサの出力を読みとるために A/D 変換ボードを使用した。使用した A/D 変換ボードはインターフェイス社の PCI-3155 である。このボードの仕様を表3. 5に示す。

表3. 5 A/D 変換ボード(PCI-3155)の仕様

入力チャンネル数	シングルエンド16チャンネル、差動8チャンネル
入力形式	マルチプレクサ方式
入力レンジ	バイポーラ : $\pm 1V$, $\pm 2.5V$, $\pm 5V$, $\pm 10V$
入力インピーダンス	$10M\Omega$ ($\pm 5\%$)
入力保護	ボードの電源ON時 : $\pm 35V$ ボードの電源OFF時 : $\pm 20V$
分解能	16ビット
AD変換器	使用素子 : AD676J相当品
相対精度	$\pm 2LSB$ (MAX) ($25^{\circ}C$)
誤差	$\pm 0.10\%$ (MAX) ($0^{\circ}C \sim 50^{\circ}C$) : $\pm 10V$ $\pm 0.15\%$ (MAX) ($0^{\circ}C \sim 50^{\circ}C$) : $\pm 1V$, $\pm 2.5V$, $\pm 5V$
変換時間	チャンネル固定時 : $10\mu s$ チャンネル切替時 (高速モード) *1 : $10\mu s$ チャンネル切替時 (通常モード) : $20\mu s$
アナログトリガ*2	トリガレベル設定値 (電圧) : 2レベル 比較電圧をDAにて設定 ($\pm 10V$ (MAX)) アナログトリガ出力 (TTL) *4 : 1点
FIFO容量	32768サンプル/ボード
対応バススロット	PCIローカルバス (Rev. 2. 1以上)



図3. 6 A/D変換ボード(PCI-3155)

3.7 ストレイン・アンプ

力センサはストレイン・ゲージを使って出力するが、それを電圧に変換するためにストレイン・アンプを使用した。使用したストレイン・アンプの仕様を表3.6に示す。

表3.6 ストレイン・アンプ仕様

実装チャンネル数	8ch
表示部	表示桁数…4桁LED 変換回数…約3回/秒 単位自動表示…(V,ACV,°C)
メモ리카ード部	シグナルコンディショナの設定値の記憶、再設定(4通り)
モニタ出力部	モニタチャンネルセレクトスイッチにより選択されたチャンネルのアナログ出力が取り出せる。 同時に、表示部に出力される。
外部インタフェース	GP-IB、RS-232C 標準装備(同時使用不可) 外部インターフェイスを利用して、ユニットの各設定、ステータスの読み出し可能(指定CH、グループ、全CH)
同期用出力	ブリッジ電源…電圧:2Vrms 周波数:25kHz又は5kHz リモート機能/全チャンネル…オートバランス ±CAL,オートレンジ、セルフチェック
適用ゲージ抵抗	120Ω～1kΩ
平衡調整方式	抵抗自動バランス
平衡調整範囲	抵抗分±約2%(±10000×10 ⁻⁶ ひずみ)
電圧感度	200×10 ⁻⁶ ひずみ入力にて5V以上(VAR最大)
測定範囲	200、500、1k、2k、5k×10 ⁻⁶ ひずみ/FS、OFF(VAR最大、BV=2V)
非直線性	±0.2%FS以内
周波数特性	DC～10kHz±10%
耐震性	3G(29.4m/s ²)
耐電圧	AC1kV1分間:コンディショナ入力端子～ケース間
使用温度湿度範囲	－10～40°C、20～85%RH以内
電源	AC100V±10%(AC120、220、240V切り換え可 約80VA DC10.5V～15V,約4A
質量	約9kg



図3. 7 ストレイン・アンプ

第4章 実験

実験全体概要図

実験装置の配置図を図4. 1に示す。

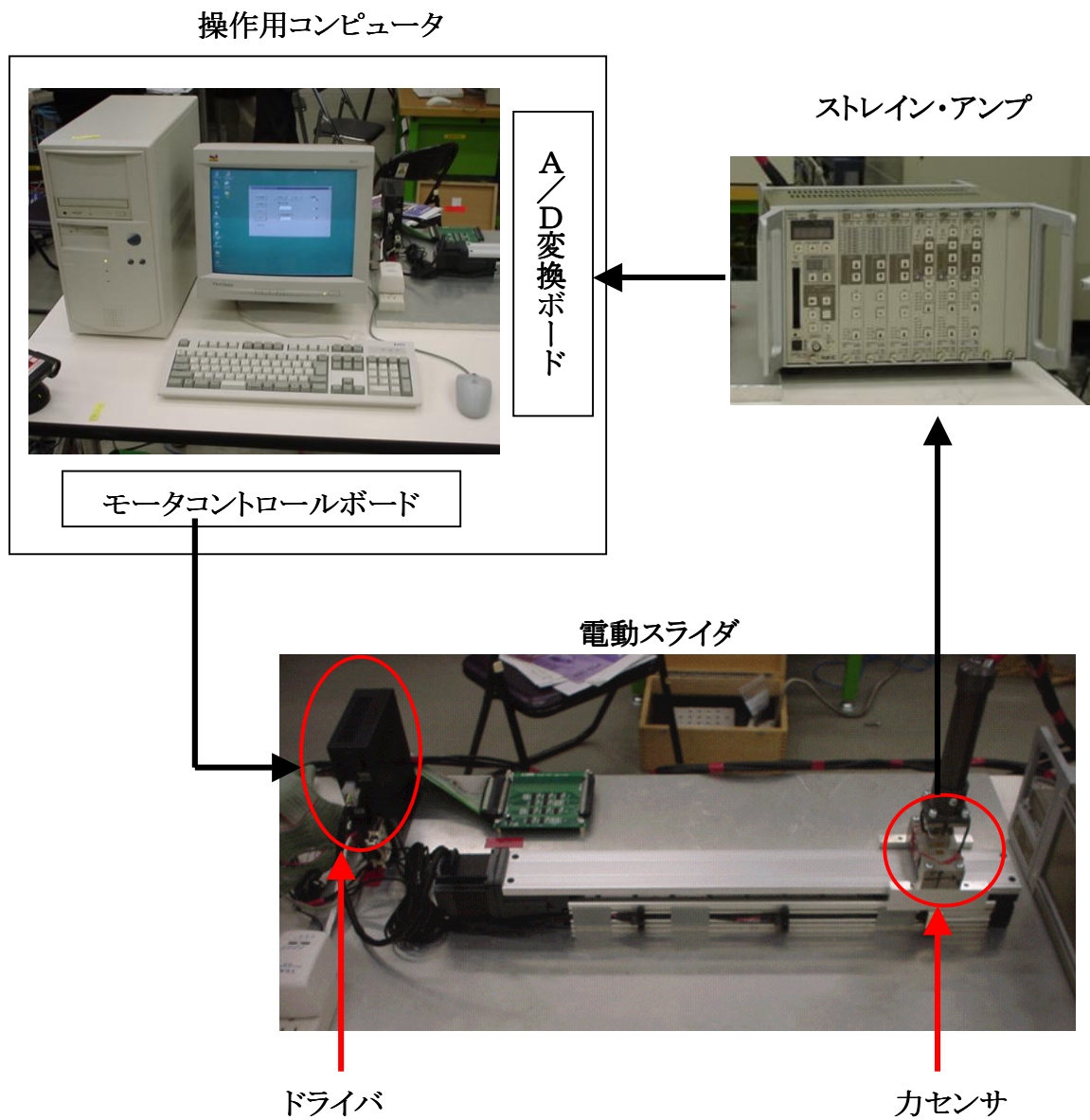


図4. 1 実験装置全体概要図

4. 1 電動スライダ基本制御実験

4. 1. 1 目的

操作用コンピュータにVisualC++を利用して操作画面を作成する。その操作画面から電動スライダを制御する。

4. 1. 2 結果と考察

図4. 2にVisualC++を利用して作成した操作画面を示す。枠(1)が実験4. 1で使
用した操作画面である。

1. 本研究では電動スライダ1軸しか使用しなかったが操作画面より軸を選択することで、X, Y, Zの3軸に対応できるようにした。
2. 移動距離・移動速度を入力し、+方向・-方向を指定することで電動スライダを動かした。入力された移動距離はパルス数に変換されドライバに送られる。ドライバから電動スライダに信号を送ることで電動スライダを動かす。
この時1パルス=0. 01mm動くことを参考にしてプログラムを作成した。
3. 電動スライダには、衝突防止のために両端にリミットセンサが取り付けられているが、なんらかの理由で非常停止しなければならなくなった時のために非常停止ボタンを作成した。

以上の点で電動スライダの基本制御は完成した。

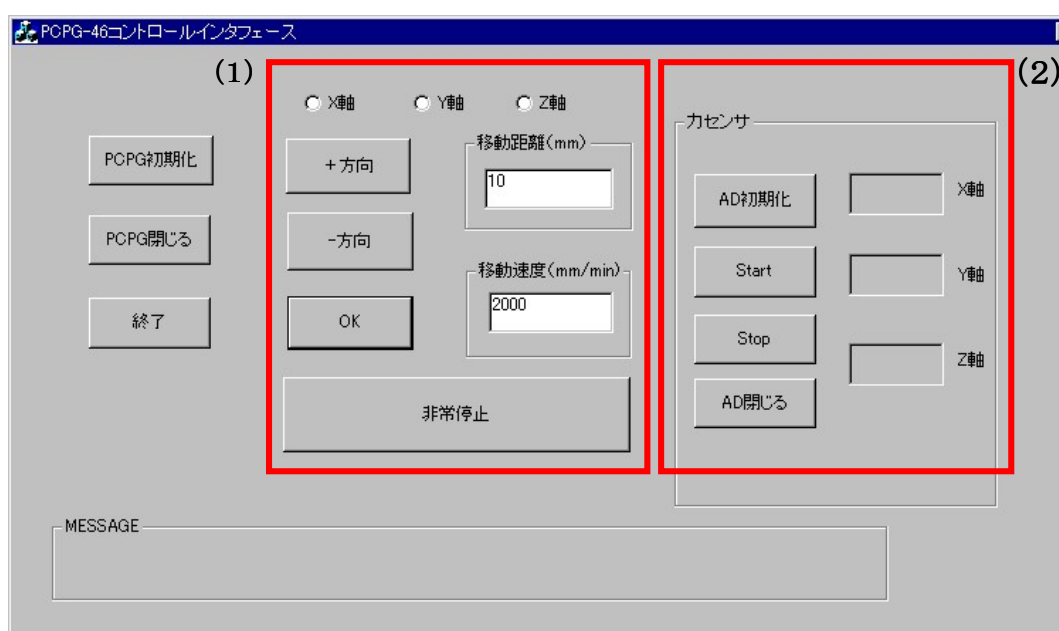


図4. 2 操作画面

力センサの検定実験

力センサが3軸力を検出するものなので、力の干渉が起きる。その干渉をなくすために非干渉化行列を求めなければならない。そのためにまず力センサの検定を行った。検定方法はX, Y, Z方向に、それぞれ2kgまでの荷重をかけその時の出力を測定するというものである。測定はストレイン・アンプを通し、A/Dコンバータを搭載した検定用コンピュータで行った。この検定用コンピュータで力を歪みに変換するマトリクスを作成し、その逆行列をMathematicaにより計算して求めた。非干渉化行列は

$$\begin{pmatrix} 0.677901 & -0.631918 & 0.646861 \\ -0.14649 & 0.731912 & -0.0117303 \\ 0.893385 & -3.24264 & -0.92493 \end{pmatrix}$$

となった。

検定実験の概要図を図4.3に示す。

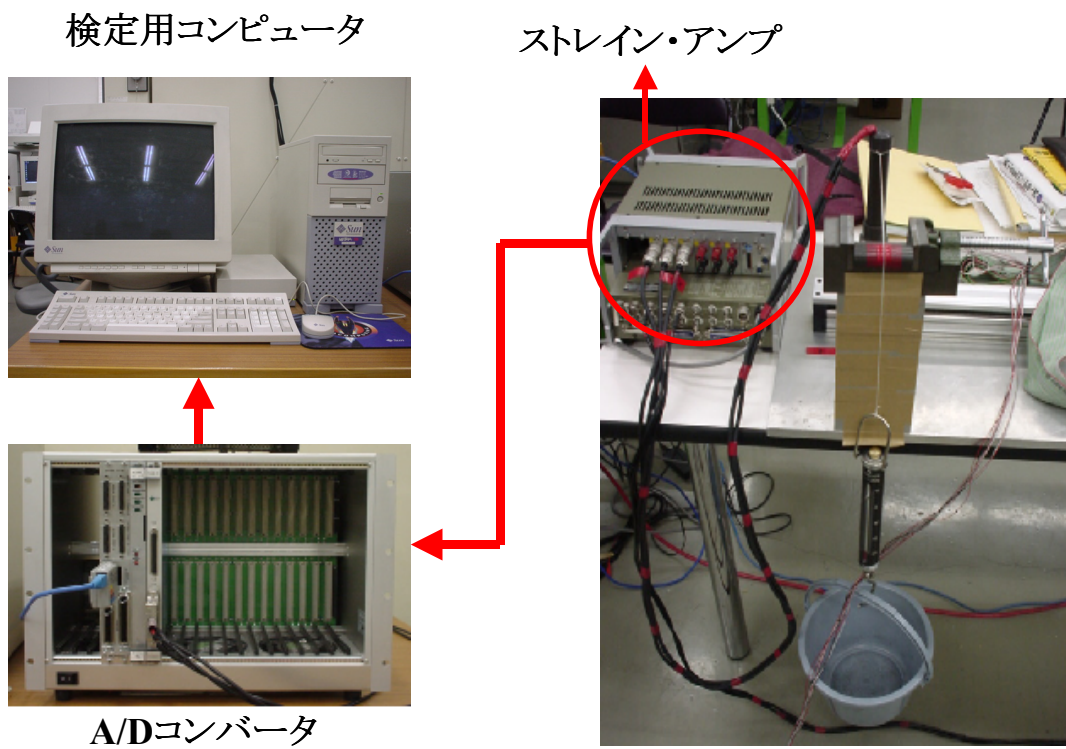


図4. 3 カセンサ検定実験概要図

4.2 力による電動スライダの制御実験

4.2.1 目的

力センサを電動スライダの上に取り付け、力を加えることで力センサからの出力をリアルタイムに変換し電動スライダに伝え制御する。この時、力センサの検定実験より求めた非干渉化行列をプログラムに組み込み実験を行う。

4.2.2 結果と考察

図4.2の枠(2)が実験4.2で使った操作画面である。

枠(2)の操作画面よりA/Dボードをオープンし、力センサより力データ(出力)を取り始める。力センサからの力データ(出力)はリアルタイムに力(N)の値で表示するようにした。

非干渉化行列を組み込んだことで力の干渉はほとんどなくなり、電動スライダは力を加えることでスムーズに動いた。

第5章

今後の展望

今後の展望

本研究では3次元ジョイスティックの基本技術を研究した。今後3次元ジョイスティックから工作機械を操作するためには、まだやらなければならないことが多数ある。

1. 本研究では1軸ずつしか動かなかったが、3軸同時に動かす必要がある。
2. 工作機械と電動スライダの可動範囲が違うのでその比率の計算が必要である
3. 本研究では力センサにどんなに大きな力を加えても電動スライダは1mmずつ、一定速度でしか動かなかったが、今後工作機械での作業をスムーズに行うためには加えた力の大きさによって移動距離・速度を変える必要がある。
4. 工作機械からの力のフィードバックを何らかの形で操作者に伝える必要がある。
5. 工作機械の現在位置や、現在の加工情報を正確に操作者に伝えることが必要である。

この他にも様々な問題点がある。これをすべて解決できたなら加工において、実際の加工の現場と操作者を切り離した快適な環境で操作できるシステムが実現するであろう。

謝辞

本研究は、長尾高明教授と李軍旗助教授の御指導を受け学び研究しました。
研究全般において、長尾高明教授、李軍旗助教授に貴重な御助言、御指導を頂きました。

同研究室の大谷さん、木崎さん、更谷さん、福見さん、増吉さん、田中さん、柳原くん、に多くの御協力を頂きました。

以上の方々に深く感謝し、厚くお礼を申し上げます。

付録

VisualC++ プログラム

```
// pcpDlg.cpp : インプリメンテーション ファイル
//
#include "stdafx.h"
#include "pcpg.h"
#include "pcpgDlg.h"
#include "pcpg46.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "windows.h"
#include "stdio.h"
#include "FbiAd.h"

#define AXIS_NUM 3

HANDLE hDeviceHandle;
ADSMPLCHREQ SmplChReq[3];
WORD wSmpData[3];

int adDataOrg[AXIS_NUM];
int adDataCal[AXIS_NUM];

double mstPerBit[AXIS_NUM];
double nonIntfMatrix[AXIS_NUM][AXIS_NUM];
double newtonPerMst[AXIS_NUM];
double newtonPerMstMatrix[AXIS_NUM][AXIS_NUM]; /* newtonPerMst *
nonIntfMatrix */
double sensorNon[AXIS_NUM][AXIS_NUM]; /* newtonPerMstMatrix *
newtonPerMst */
double meanForce[AXIS_NUM];
int counter;

char dummy[256];

pcpg46 pcpDg;
```

```

TCHAR          szBuf[1024];
BYTE           bData[256];
WORD           wBsn, wAxis;
PCPG46RESOURCE ri;
BYTE           b;
DWORD          dwCount;

////////////////////////////////////
// アプリケーションのバージョン情報で使われている CAboutDlg ダイアログ

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// ダイアログ データ
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    }}AFX_DATA

// ClassWizard は仮想関数のオーバーライドを生成します
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV のサポート
    }}AFX_VIRTUAL

// インプリメンテーション
protected:
   //{{AFX_MSG(CAboutDlg)
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    }}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
}

```

```

        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
        // メッセージ ハンドラがありません。
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPcpgDlg ダイアログ

CPcpgDlg::CPcpgDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPcpgDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPcpgDlg)
        m_distance = 0;
        m_feedrate = 0;
        m_fx = _T("");
        m_fy = _T("");
        m_fz = _T("");
    //}}AFX_DATA_INIT
    // メモ: LoadIcon は Win32 の DestroyIcon のサブシーケンスを要求しません。
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPcpgDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPcpgDlg)
        DDX_Text(pDX, IDC_EDIT_DISTANCE, m_distance);
        DDV_MinMaxInt(pDX, m_distance, 1000, 50000);
        DDX_Text(pDX, IDC_EDIT_FEED, m_feedrate);
        DDX_Text(pDX, IDC_FORCEX, m_fx);
        DDX_Text(pDX, IDC_FORCEY, m_fy);
        DDX_Text(pDX, IDC_FORCEZ, m_fz);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPcpgDlg, CDialog)
   //{{AFX_MSG_MAP(CPcpgDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP

```

```

ON_BN_CLICKED(IDC_PCPGINIT, OnPcpginit)
ON_BN_CLICKED(IDC_MINUS, OnMinus)
ON_BN_CLICKED(IDC_PLUS, OnPlus)
ON_BN_CLICKED(IDC_PCPGCLOSE, OnPcpgclose)
ON_EN_CHANGE(IDC_EDIT_DISTANCE, OnChangeEditDistance)
ON_EN_CHANGE(IDC_EDIT_FEED, OnChangeEditFeed)
ON_BN_CLICKED(IDC_AD_START, OnAdStart)
ON_WM_TIMER()
ON_BN_CLICKED(IDC_ADINIT, OnAdinit)
ON_BN_CLICKED(IDC_AD_STOP, OnAdStop)
ON_BN_CLICKED(IDC_ADCLOSE, OnAdclose)
ON_EN_CHANGE(IDC_EDIT_MESSAGE, OnChangeEditMessage)
ON_EN_CHANGE(IDC_FORCEX, OnChangeForcex)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPcpgDlg メッセージ ハンドラ

BOOL CPcpgDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // "バージョン情報..." メニュー項目をシステム メニューへ追加します。

    // IDM_ABOUTBOX はコマンド メニューの範囲でなければなりません。
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING,    IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // このダイアログ用のアイコンを設定します。フレームワークはアプリケーションのメイン
    // ウィンドウがダイアログでない時は自動的に設定しません。

```

```

        SetIcon(m_hIcon, TRUE); // 大きいアイコンを設定
        SetIcon(m_hIcon, FALSE); // 小さいアイコンを設定

// TODO: 特別な初期化を行う時はこの場所に追加してください。

return TRUE; // TRUE を返すとコントロールに設定したフォーカスは失われません。
}

void CPcpgDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFFF) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// もしダイアログボックスに最小化ボタンを追加するならば、アイコンを描画する
// コードを以下に記述する必要があります。MFC アプリケーションは document/view
// モデルを使っているので、この処理はフレームワークにより自動的に処理されます。

void CPcpgDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // 描画用のデバイス コンテキスト

        SendMessage(WM_ICONERASEBKGND, (LPARAM) dc.GetSafeHdc(),
0);

        // クライアントの矩形領域内の中央
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // アイコンを描画します。
        dc.DrawIcon(x, y, m_hIcon);
    }
}

```

```

    }
    else
    {
        CDialog::OnPaint();
    }
}

```

// システムは、ユーザーが最小化ウィンドウをドラッグしている間、
// カーソルを表示するためにここを呼び出します。

```

HCURSOR CPcpgDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

```

```

void CPcpgDlg::OnPcpginit()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    //-----
    // Pcp46.Dll を開く
    //-----
    if ( FALSE == pcpg.Pcpg46wDllOpen() ) {
        pcpg.ErrorMessage(PCPG46_BSN_AUTO);
        return ;          // DLL のロードまたはドライバのオープンに失敗しました
    }
    //-----
    // DLLバージョン情報の表示
    //-----
    if ( FALSE == pcpg.Pcpg46wGetLibVersion(bData) ) {
        pcpg.ErrorMessage(PCPG46_BSN_AUTO);
        return ;
    }
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T("Dll Version %s"),
        bData );
    pcpg.ResultMessage( szBuf );
    //-----
    // ドライババージョン情報の表示
    //-----
    if ( FALSE == pcpg.Pcpg46wGetDrvVersion(bData) ) {
        pcpg.ErrorMessage(PCPG46_BSN_AUTO);
        return ;
    }
    wsprintf( szBuf,

```

```

        _T("PCPG46¥n")
        _T("Drv Version %s"),
        bData );
pcpg.ResultMessage( szBuf );
//-----
// デバイスの使用を宣言する
//-----
wBsn = PCPG46_BSN_AUTO;           // 空きデバイス自動検索を指定
if ( FALSE == pcpg.Pcpg46wCreate(&wBsn) ){
    pcpg.ErrorMessage(PCPG46_BSN_AUTO);
    return ;
}
wsprintf( szBuf,
        _T("PCPG46 Created¥n")
        _T("BSN Number %d"), wBsn );
pcpg.ResultMessage( szBuf );
//-----
// リソース情報を表示
//-----
if ( FALSE == pcpg.Pcpg46wGetResource(wBsn,&ri)){
    pcpg.ErrorMessage(wBsn);
    return ;
}
wsprintf(szBuf, _T("Board  Name: %s¥n")
        _T("IO Address : %04Xh-%04Xh¥n"),
        szPcpg46,
        ri.dwIOPortBase[1],ri.dwIOPortBase[1]      +
ri.dwIOPortLength[1] - 1 );
pcpg.ResultMessage(szBuf);

//-----
// 第1軸を選択
//-----
wAxis = PCPG46_AXIS_1;
//-----
// 制御軸の初期化
//-----
if ( FALSE == pcpg.Pcpg46_InitAxis(wBsn,wAxis)){
    pcpg.ErrorMessage(wBsn);
    return ;
}
wsprintf( szBuf,
        _T("PCPG46¥n")

```

```

        _T("制御軸 %d を初期化しました"), wAxis );
pcpg.ResultMessage( szBuf );

}

void CPcpgDlg::OnMinus()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

    //-----
    // 一方向に 100*m_distance パルス出力
    //-----
    if ( FALSE == pcpg.Pcpg46wDataFullWrite(
        wBsn,        wAxis,        PCPG46_MINUS_PRESET_PULSE_DRIVE,
100*m_distance ) ) {
        pcpg.ErrorMessage(wBsn);
        return ;
    }
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T("一方向に%d mm 移動しています"), m_distance);
    pcpg.ResultMessage( szBuf );

}

void CPcpgDlg::OnPlus()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

    //-----
    // +方向に 100*m_distance パルス出力
    //-----
    if ( FALSE == pcpg.Pcpg46wDataFullWrite(
        wBsn,        wAxis,        PCPG46_PLUS_PRESET_PULSE_DRIVE,
100*m_distance ) ) {
        pcpg.ErrorMessage(wBsn);
        return ;
    }
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T("+方向に%d mm 移動しています"),m_distance);
    pcpg.ResultMessage( szBuf );
}

```

```

}

void CPcpgDlg::OnPcpgclose()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

    //-----
    // デバイスを解放する
    //-----
    if ( FALSE == pcpg.Pcpg46wClose(wBsn)){
        pcpg.ErrorMessage(wBsn);
        return ;
    }
    wsprintf( szBuf,
        _T("PCPG46 Close¥n")
        _T("Bsn %d"), wBsn );
    pcpg.ResultMessage( szBuf );
    //-----
    // Pcpg46.Dll を閉じる
    //-----
    if ( FALSE == pcpg.Pcpg46wDllClose() ) {
        pcpg.ErrorMessage(PCPG46_BSN_AUTO);
        return ;
        // DLL のアンロード
に失敗しました
    }
    return ;
}

void CPcpgDlg::OnChangeEditDistance()
{
    // TODO: これが RICHEDIT コントロールの場合、コントロールは、IParam マスク
    // 内での論理和の ENM_CHANGE フラグ付きで CRichEditCtrl0.SetEventMask()
    // メッセージをコントロールへ送るために CDialog::OnInitDialog() 関数をオーバー
    // ライドしない限りこの通知を送りません。

    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    CEdit * str = (CEdit *) GetDlgItem(IDC_EDIT_DISTANCE);
    char count[10];
    int n;
    n=str->GetLine(0,count,10);
    count[n] = 0;
    m_distance = atoi(count);
}

```

```

    }

void CPcpgDlg::OnChangeEditFeed()
{
    // TODO: これが RICHEDIT コントロールの場合、コントロールは、IParam マスク
    // 内での論理和の ENM_CHANGE フラグ付きで CRichEditCtrl0.SetEventMask()
    // メッセージをコントロールへ送るために CDialog::OnInitDialog() 関数をオーバー
    // ライドしない限りこの通知を送りません。

    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    CEdit * str = (CEdit *) GetDlgItem(IDC_EDIT_FEED);
    char count[10];
    int n;
    n=str->GetLine(0,count,10);
    count[n] = 0;
    m_feedrate = atoi(count);

    if ( FALSE == pcpg.Pcpg46_Change_Feed(wBsn,wAxis,m_feedrate)){
        pcpg.ErrorMessage(wBsn);
        return ;
    }
    /*
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T(" 制 御 軸  %d  の 送 り 速 度 を %d  mm/min に 変 換 し ま し た "),
wAxis,m_feedrate);
    pcpg.ResultMessage( szBuf );
    */
}

// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

void CPcpgDlg::OnAdStart()
{
    int axis;

    counter = 0;
    for (axis = 0; axis < AXIS_NUM; axis++)
        meanForce[axis] = 0.0;
    SetTimer(1,5,NULL);
}

```

```

// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

void CPcpgDlg::OnOK()
{
    // TODO: この位置にその他の検証用のコードを追加してください

    CDialog::OnOK();
}

void CPcpgDlg::OnTimer(UINT nIDEvent)
{
    int axis, index;
    double force[AXIS_NUM];

    AdInputAD(hDeviceHandle, 3, AD_INPUT_SINGLE, &SmplChReq[0],
&wSmpData);
    for (axis = 0; axis < AXIS_NUM; axis++)
        force[axis] = 0.0;
    for (axis = 0; axis < AXIS_NUM; axis++) {
        for (index = 0; index < AXIS_NUM; index++) {
            force[axis]
sensorNon[axis][index]*((int)wSmpData[index]-adDataOrg[index]);
        }
    }
    for (axis = 0; axis < AXIS_NUM; axis++)
        meanForce[axis]+=force[axis];
    counter++;
    if(counter == 10){
        for (axis = 0; axis < AXIS_NUM; axis++)
            meanForce[axis] = meanForce[axis]/10.0;
        sprintf(dummy, "%lf",meanForce[0]);
        m_fx=dummy;
        sprintf(dummy, "%lf",meanForce[1]);
        m_fy=dummy;
        sprintf(dummy, "%lf",meanForce[2]);
        m_fz=dummy;
        UpdateData(false);
        if (meanForce[0] >= 8.0){
            if ( FALSE == pcpg.Pcpg46wDataFullWrite(
                wBsn, wAxis, PCPG46_PLUS_PRESET_PULSE_DRIVE,

```

```

100*m_distance)) {
    pcpg.ErrorMessage(wBsn);
    return ;
}
}
else
if (meanForce[0] <= -8.0){
if ( FALSE == pcpg.Pcpg46wDataFullWrite(
    wBsn,      wAxis,      PCPG46_MINUS_PRESET_PULSE_DRIVE,
100*m_distance)) {
    pcpg.ErrorMessage(wBsn);
    return ;
}
}
counter = 0;
for (axis = 0; axis < AXIS_NUM; axis++)
meanForce[axis] = 0.0;
}

```

// TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください

```

CDialog::OnTimer(nIDEvent);
}

```

/* 非干渉化行列のセット*/

```

void CPcpgDlg::SetMatrix()
{

```

```

    int axis, index;

```

```

    nonIntfMatrix[0][0] = 1.03371;
    nonIntfMatrix[0][1] = -0.725237;
    nonIntfMatrix[0][2] = 0.186736;
    nonIntfMatrix[1][0] = -0.148806;
    nonIntfMatrix[1][1] = 0.722155;
    nonIntfMatrix[1][2] = 0.012234;
    nonIntfMatrix[2][0] = 1.10853;
    nonIntfMatrix[2][1] = -0.34993;
    nonIntfMatrix[2][2] = -1.28004;

```

```

    newtonPerMst[0] = 1.0;
    newtonPerMst[1] = 1.0;
    newtonPerMst[2] = 1.0;
    for (axis = 0; axis < AXIS_NUM; axis++) {
        for (index = 0; index < AXIS_NUM; index++) {

```

```

        newtonPerMstMatrix[axis][index] =
            newtonPerMst[axis] * nonIntfMatrix[axis][index];
    }
}

} /* SetMatrix0 */

/* 0mst. での A/D の出力を測定する, 結果は adDataOrg に格納 */
void CPcpgDlg::GetOrgAD()
{
    static int meani = 10;          /* meani 回の平均を取る*/
    int  mi, axis;
    int    tmpDataOrg[AXIS_NUM];

    /* tmpDataOrg の初期化 */
    for (axis = 0; axis < AXIS_NUM; axis++) tmpDataOrg[axis] = 0;
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T("Balance Strain Amp. and Hit Return"));
    pcpg.ResultMessage( szBuf );

    /* Sample the data from each axis of AD converter */
    /* Calculate the sum of each axis */
    SmplChReq[0].ulChNo = 1;
    SmplChReq[0].ulRange = AD_5V;
    SmplChReq[1].ulChNo = 2;
    SmplChReq[1].ulRange = AD_5V;
    SmplChReq[2].ulChNo = 3;
    SmplChReq[2].ulRange = AD_5V;

    for (mi = 0; mi < meani; mi++) {
        AdInputAD(hDeviceHandle,    3,    AD_INPUT_SINGLE,    &SmplChReq[0],
&wSmpData);
        for (axis = 0; axis < AXIS_NUM; axis++)
            tmpDataOrg[axis] = tmpDataOrg[axis] +(int)wSmpData[axis];
        Sleep(100);
    }
    /* Calculate the avarage of each axis and Print them */
    for (axis = 0; axis < AXIS_NUM; axis++) {
        adDataOrg[axis] = (int)(tmpDataOrg[axis] / meani);
    }
    sprintf(dummy, "%d",adDataOrg[0]);
    m_fx=dummy;
    sprintf(dummy, "%d",adDataOrg[1]);

```

```

        m_fy=dummy;
        sprintf(dummy, "%d",adDataOrg[2]);
        m_fz=dummy;
        UpdateData(false);
    } /* GetOrgAD0 */

/* 100mst.での A/D の出力を測定する, 結果は adDataCal に格納*/
void CPcpgDlg::GetCalAD0
{
    static int meani = 10;          /* meani 回の平均を取る*/
    int  mi, axis;
    int  tmpDataCal[AXIS_NUM];

    /* tmpDataCal の初期化*/
    for (axis = 0; axis < AXIS_NUM; axis++) tmpDataCal[axis] = 0;
    wsprintf( szBuf,
        _T("PCPG46¥n")
        _T("Turn on 100 micro strain calibration switch and Hit Return"));
    pcpg.ResultMessage( szBuf );

    SmplChReq[0].ulChNo = 1;
    SmplChReq[0].ulRange = AD_5V;
    SmplChReq[1].ulChNo = 2;
    SmplChReq[1].ulRange = AD_5V;
    SmplChReq[2].ulChNo = 3;
    SmplChReq[2].ulRange = AD_5V;

    for (mi = 0; mi < meani; mi++) {
        AdInputAD(hDeviceHandle,    3,    AD_INPUT_SINGLE,    &SmplChReq[0],
        &wSmpData);
        for (axis = 0; axis < AXIS_NUM; axis++)
            tmpDataCal[axis] = tmpDataCal[axis] +(int)wSmpData[axis];
        Sleep(100);
    }

    for (axis = 0; axis < AXIS_NUM; axis++) {
        adDataCal[axis] = (int)(tmpDataCal[axis] / meani);
    }
    sprintf(dummy, "%d",adDataCal[0]);
    m_fx=dummy;
    sprintf(dummy, "%d",adDataCal[1]);
    m_fy=dummy;
    sprintf(dummy, "%d",adDataCal[2]);

```

```

        m_fz=dummy;
        UpdateData(false);
        wsprintf( szBuf,
            _T("PCPG46¥n")
            _T("Turn off Cal and Hit return¥n"));
        pcpg.ResultMessage( szBuf );

} /* GetCalAD0 */

/* calibration の結果より1ビット当たりの歪の計算*/
void CPcpgDlg::CalcMstPerBit()
{
    int axis;

    /* 1ビット当たりの歪の計算(100ust の場合) */
    for (axis = 0; axis < AXIS_NUM; axis++) {
        mstPerBit[axis]
            = (double)100.0 / (adDataCal[axis] - adDataOrg[axis]);
    }
    sprintf(dummy, "%lf",mstPerBit[0]);
    m_fx=dummy;
    sprintf(dummy, "%lf",mstPerBit[1]);
    m_fy=dummy;
    sprintf(dummy, "%lf",mstPerBit[2]);
    m_fz=dummy;
    UpdateData(false);

} /* CalcMstPerBit */

/* newtonPerMstMatrix × mstPerBit = sensorNon を計算する*/
void CPcpgDlg::CalcSensorNon()
{
    int axis, index;

    /* 非干渉化行列×1ビット当たりの歪の計算*/
    for (axis = 0; axis < AXIS_NUM; axis++) {
        for (index = 0; index < AXIS_NUM; index++) {
            sensorNon[axis][index]
                = newtonPerMstMatrix[axis][index] * mstPerBit[axis];
        }
    }
} /* CalcSensorNon */

```

```

void CPcpgDlg::OnAdinit()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    hDeviceHandle = AdOpen("FBIAD1");
    if (hDeviceHandle == INVALID_HANDLE_VALUE){
        MessageBox("Device FBIAD1 は使用できません");
    }
    else{
        MessageBox("AD ボードオープンしました");
    }
    SetMatrix();
    GetOrgAD();
    GetCalAD();
    CalcMstPerBit();
    CalcSensorNon();
}

void CPcpgDlg::OnAdStop()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    KillTimer(1);
    // AdClose(hDeviceHandle);
}

void CPcpgDlg::OnAdclose()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    AdClose(hDeviceHandle);
}

void CPcpgDlg::OnChangeEditMessage()
{
    // TODO: これが RICHEDIT コントロールの場合、コントロールは、IParam マスク
    // 内での論理和の ENM_CHANGE フラグ付きで CRichEditCtrl0.SetEventMask()
    // メッセージをコントロールへ送るために CDialog::OnInitDialog() 関数をオーバー
    // ライドしない限りこの通知を送りません。

    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
}

```

```
void CPcpgDlg::OnChangeForcex()
{
    // TODO: これが RICHEDIT コントロールの場合、コントロールは、IParam マスク
    // 内での論理和の ENM_CHANGE フラグ付きで CRichEditCtrl0.SetEventMask()
    // メッセージをコントロールへ送るために CDialog::OnInitDialog() 関数をオーバー
    // ライドしない限りこの通知を送りません。

    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
}
```